OPTIMISATION DE PROCESSUS DECISIONNELS POUR LA ROBOTIQUE

Malik Ghallab

▶ **To cite this version:**

Malik Ghallab. OPTIMISATION DE PROCESSUS DECISIONNELS POUR LA ROBOTIQUE. Automatique / Robotique. Université Paul Sabatier - Toulouse III, 1982. Français. NNT : . tel-00514173

HAL Id: tel-00514173

https://theses.hal.science/tel-00514173

Submitted on 1 Sep 2010

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

N° d'ordre : 1065

LAAS

THESE

de Doctorat d'Etat

présentée à

L'UNIVERSITE PAUL SABATIER DE TOULOUSE

pour obtenir le grade de

DOCTEUR es-SCIENCES

Algorithmique - Robotique

par

Malik GHALLAB

OPTIMISATION DE PROCESSUS DECISIONNELS POUR LA ROBOTIQUE

Soutenue le 28 octobre 1982, devant la Commission d'Examen :

MM. Georges GIRALT — *Président*

Marc COURVOISIER
Henri FARRENY
Hervé GALLAIRE
Gérard GUIHO
Richard M. KARP
Maurice NIVAT
Roland PRAJOUX

GHALLAB(Malik) - Optimisation de Processus Décisionnels pour la Robotique. - p.439

Th. Doctorat d'Etat : Robotique : Toulouse III : 1982 ; 1065

RESUME :

A partir du formalisme des systèmes de règles de décision, on définit deux types de processus décisionnels : les processus fermés (PDF) portant sur des systèmes représentés dans des espaces finis ; et les processus ouverts (PDO) pour des systèmes à espaces d'états infinis.
On considère ces processsus comme des algorithmes particuliers et on s'intéresse à leur modélisation, leur analyse et l'optimisation de leur complexité, selon différents critères, en tenant compte de la complexité de la tâche d'optimisation elle-même. La caractérisation de cette tâche, en tant que problème NP-dur au sens fort et approximation Np-dur, conduit à développer des schémas d'approximation qui généralisent les algorithmes de recherche heuristique dans les graphes et hypergraphes en procédures ε -admissibles.
Deux processus décisionnels en robotique sont traités :
l'un fermé portant sur l'apprentissage d'un classifieur pour l'identification d'objets, et l'autre ouvert pour la génération de plans.

MOTS CLES : Robotique, Processus Décisionnels, Systèmes de Règles de Décision, Optimisation Combinatoire, Recherche Heuristique, Complexité Algorithmique, NP-Complétude

JURY et date de soutenance : 28 Octobre 1982

Président : G. GIRALT

Membres : M. COURVOISIER
H. FARRENY
H. GALLAIRE
G. GUIHO
R. M. KARP
M. NIVAT
R. PRAJOUX

DEPOT à la Bibliothèque Universitaire en quatre exemplaires

Abstract :

On the basis of the general formalism of "Production Systems", two types of Decision Processes are defined : closed processes for finite state-space systems, and Open processes for infinite state space systems.

The thesis considers such processes as particular algorithms, and deals with their modelization, analysis and optimization of their complexity for various criteria and cost models, taking into account the complexity of the optimization task itself. This task is characterized as NP-hard in the strong sense, and carried out by some original approximation schemes, developped as a generalisation of admissible heuristic search procedures to near admissible ones.

Two Robotics decision processes are considered : synthesis of optimal classifiers for object recognition, and plan generation.

Key words : Robotics, Production Systems, Heuristic Search, Algoritmic Complexity, NP-Completeness

UNIVERSITE PAUL SABATIER

PRESIDENCE

M. BANCEL ..... Président
M. MADAULE ..... Vice-Président - Sciences
.....

ORDRE DES SCIENCES

HONORARIAT

M. AGID ..... Professeur honoraire
M. BEDOS ..... Professeur honoraire
M. BLAIZOT ..... Doyen honoraire
M. CAPDECOMME .. Doyen honoraire, Recteur honoraire, Correspondant de l'Institut, Professeur honoraire
Mlle De FERRE ..... Professeur honoraire
M. DUPOUY ..... Membre de l'Institut, Doyen honoraire, Directeur honoraire du C.N.R.S. Professeur honoraire
M. DURAND Emile .. Doyen honoraire, Professeur honoraire
M. FERT ..... Professeur honoraire
M. GALLAIS ..... Professeur honoraire, Membre de l'Institut
M. LAFOURCADE .. Professeur honoraire
M. LESBRE ..... Professeur honoraire
M. MARGULIS ..... Professeur honoraire
M. MASDUPUY ..... Professeur honoraire
M. MATHIS ..... Doyen honoraire
M. MIGNONAC ..... Professeur honoraire
M. MORQUER ..... Professeur honoraire, Correspondant de l'Institut
M. ORLIAC ..... Professeur honoraire
M. PERRIER ..... Professeur honoraire
M. SECONDAT ..... Professeur honoraire
M. SERFATY ..... Professeur honoraire
M. TEISSIE-SOLIER . Professeur honoraire
M. TRICHE ..... Professeur honoraire

EMERITAT

M. AGID
M. CAPDECOMME
Mlle De FERRE
M. FERT
M. GALLAIS
M. LESBRE
M. ORLIAC
M. SERFATY

CORPS ENSEIGNANT

PROFESSEURS DE CLASSE EXCEPTIONNELLE ET DE 1ère CLASSE

M. HURON ..... Mathématiques Appliquées
M. LEDOUX ..... Zoologie Appliquée
M. MATHIS ..... Chimie
M. ANGELIER ..... Zoologie
M. FARRAN ..... Minéralogie et Géotechnique
M. LAUDET ..... Physique Théorique et Calcul Numérique
M. LAGASSE ..... Electrotechnique
M. BLANC ..... Physique Nucléaire
M. LEREDDE ..... Botanique
M. LELUBRE ..... Géologie
M. LALAGUE ..... Mathématiques Générales
M. BOUIGUE ..... Astronomie
M. ASSELINEAU .... Chimie Biologique
M. MAURET ..... Chimie Systématique
M. MONTANT ..... Cryptogamie
M. GAUTIER ..... Physique
M. CRUMEYROLLE . Mathématiques
M. GOURINARD .... Géologie
M. PULOU ..... Minéralogie
M. CAMBOU ..... Physique Spatiale
M. LACOSTE ..... Electrotechnique
M. THIBAULT ..... Mécanique Rationnelle et Appliquée
M. MASCART ..... Mathématiques
M. MEDIONI ..... Psychophysiologie
M. RAYNAUD P. .... Physiologie Animale
M. ZALTA ..... Chimie Biologique
M. SEVELY ..... Electrotechnique
M. POMMIEZ ..... Mathématiques
M. REY Paul ..... Biologie Végétale
M. COULOMB ..... Physique
M. TRINQUIER ..... Physique
M. MARONI ..... Chimie
M. BEETSCHEN .... Biologie Générale
M. DERACHE ..... Physiologie Animale
M. SATGE ..... Chimie Organique
M. LATTES ..... Chimie
M. VEDRENNE ..... Géophysique
M. DURAND-DELGA Géologie, Correspondant de l'Institut
M. CARRARA ..... Physique
M. MAHENC ..... Chimie
M. MIROUSE ..... Géologie
M. BITSCH ..... Zoologie
M. DEGEILH ..... Physique
M. MARTIN J.C. .... Génie Electrique
M. REY Gérard ..... Génie Electrique
M. SICARD ..... Biologie Génétique
M. SOUQUET ..... Géologie
M. TOUZE ..... Physiologie Végétale
M. FRASNAY ..... Mathématiques (Algèbre et Combinatoire)
M. CASSAGNAU .... Zoologie
M. CAUSSINUS ..... Mathématiques Appliquées (Statistiques Appliquées)
M. PESCIA ..... Physique
M. PICCA ..... Physique de l'Atmosphère
M. BAUDIERE ..... Botanique Fondamentale et Pyrénéenne
M. BARRANS ..... Chimie Physique Organique
M. POILBLANC ..... Chimie Minérale
M. PERENNOU ..... Informatique
M. ATTEIA ..... Mathématiques
M. CASTAN ..... Informatique
M. COLLETTE ..... Physique
M. REME ..... Mesures Physiques
M. CUPPENS ..... Mathématiques
M. BAUDRAS ..... Chimie Biologique
M. BEAUFILS ..... Informatique
M. CALVET ..... Mécanique des Fluides
M. LETAC ..... Mathématiques
M. BOUDET ..... Physiologie Végétale

PROFESSEURS DE 2ème CLASSE

M. MERIC	Mathématiques Appliquées
Mme LECAL	Zoologie
M. PILOD	Physique
M. LARROQUE	Physique
Mme LAUDET-LAPEYRE	Mathématiques - Informatique
M. BERTRAND	Chimie
M. DESQ	Mathématiques
M. ROCARD	Electronique
M. GUERIN	Mathématiques
M. SCHNEIDER	Biologie Cellulaire
M. de LOTH	Chimie Physique
M. SAPORTE	Physique
M. THENOZ	Génie Civil
M. DURAND Ph.	Physique
M. FONTAN	Physique Nucléaire
M. PAGANI	Physique
M. BERTHELEMY	Zoologie
M. TERJANIAN	Mathématiques
M. MORUCCI	Génie Biologique et Médical
M. SOTIROPOULOS	Chimie Organique
M. VERDIER	Physique
M. ETTINGER	Mathématiques
M. BONNET Louis	Biologie
M. JOSSERAND	Mesures Physiques
M. ROUTIE	Génie Chimique
M. COTTU	Génie Mécanique
M. HURAUX	Physique
Mme GERVAIS	Chimie Inorganique
M. BANCEL	Mathématiques
M. LOUARN	Génétique
M. HERAULT	Chimie
M. GRANDET	Génie Civil
Mlle BARBANCE	Mathématiques
M. GILLY	Génie Mécanique
M. MARAL	Mesures Physiques
M. LEGRAND	Génie Civil
M. ABATUT	Electronique, Electrotechnique, Automatique
M. MAUSS	Mécanique
M. BETOURNE	Informatique
M. CAMPAN	Psychophysiologie
M. CLERC	Mécanique
M. GRIFONE	Mathématiques
M. COUOT	Mathématiques, Analyse Numérique
M. NGUYEN THANH VAN	Mathématiques
M. TRAVERSE	Problèmes Chimiques de l'Energie
M. ALRAN	Génie Chimique
M. REY J.	Géologie Sédimentaire et Paléontologie
M. DARTIGUENAVE	Chimie Minérale Moléculaire
M. PRADINES	Mathématiques
M. GALINIER	Informatique
M. VIGNOLLE	Informatique
M. DEPARIS	Embryologie
M. CAVALIE	Physiologie Végétale
M. MASSOL	Chimie des Composés Organiques et Organominéraux d'intérêt biologique
M. HARTMANN	Mécanique
M. ROUSSET	Chimie Appliquée (Matériaux)
M. HOLLANDE	Biologie Cellulaire
M. DUGAS	Physique des Energies Nouvelles
M. BENOIT-CATTIN	Physique
M. COMTAT	Chimie Appliquée
M. LANEELLE	Biochimie
M. LUGUET	Informatique Fondamentale et Appliquée
M. BONNET J.J.	Chimie Minérale
M. PERAMI	Minéralogie et Matériaux
M. AUDOUNET	Mathématiques
M. PERIE	Chimie Organique
M. AMBID	Physiologie
M. AURIOL	Biologie
M. COURVOISIER	Electronique, Electrotechnique, Automatique
M. FORTUNE	Géologie
Mlle RIVIERE	Chimie
M. TIRABY	Biologie
M. BARONNET	Thermodynamique Energétique
M. COMBES	Génie Electrique
M. DUBAC	Mesures Physiques
M. GUMOWSKI	Mécanique
M. RAUGI	Mathématiques
M. HIRIART-URRUTY	Mathématiques
M. BALLADORE	Electronique et Electrotechnique
M. HARAN	Chimie Minérale
M. RENUCCI	Physique du Solide et Cristallographie
Mme VAUCLAIR	Astronomie, Physique Spatiale, Géophysique
M. COLOMBEL	Zoologie et Ecologie
M. DURRIEU	Biologie
M. SAINT-MARC	Mesures Physiques
M. ROQUES	Génie Mécanique
M. CORDIER	Génie Mécanique
M. JOFFROY	Génie Electrique

PROFESSEURS ASSOCIES

M. COLLINS	Génie Mécanique
M. WALLACE	Physique

CHERCHEURS DU C.N.R.S.

DIRECTEURS DE RECHERCHE

M. ESTEVE Daniel
M. GALY Jean
M. GIRALT Georges
M. LABARRE Jean-François
M. LAURENT Jean-Pierre
M. LEGRIS
M. MARTINOT Henri
M. MAZEROLLES
M. PRADAL
M. WOLF Robert

MAITRES DE RECHERCHE

M. AGUILAR-MARTIN José
Mme ASSELINEAU Cécile
M. AYROLES René
M. AZEMA Pierre
Mme BENAZETH Nicole
M. BUXO Jean
Mme DARTIGUENAVE M.
Mme DUPRAT Anne-Marie
M. HAWKES Peter
M. HOUALLA Doureid
M. JEREBZOFF
M. MALRIEU J.P.
Mme MARONI Yvette
Mme MATHIS
M. MUNOZ Aurélio
M. NAVECH
M. PRAJOUX Roland
M. SEVELY Jean
M. VACQUIE Serge

CORPS DES OBSERVATOIRES ASTRONOMIQUES ET INSTITUTS DE PHYSIQUE DU GLOBE

Mme ANDRILLAT Y.	Astronome titulaire
M. AZZOPARDI M.	Astronome adjoint
M. COUPINOT G.	Astronome adjoint
M. LEROY J.L.	Astronome adjoint
M. MIANES	Astronome adjoint
M. MULLER R.	Astronome adjoint
M. PEDOUSSAUT A.	Astronome adjoint
M. QUERCI F.	Astronome adjoint
M. ROBLEY R.	Physicien titulaire
M. ROZELOT J.P.	Physicien adjoint
M. SAISSAC J.	Physicien titulaire
M. ZAHN J.P.	Astronome titulaire

ADMINISTRATION

M. PRINEAU	Secrétaire Général de l'Université

INSTITUT NATIONAL POLYTECHNIQUE DE TOULOUSE

PRESIDENCE

M. NOUGARO ..... Président
M. MARTY ..... Vice-Président
M. ENJALBERT .... Vice-Président
M. ANDRE ..... Vice-Président
M. CONSTANT ..... Vice-Président
M. MONTEL ..... Président d'honneur

HONORARIAT

M. BIREBENT ..... Professeur honoraire
M. CASTAGNETTO .. Professeur honoraire
M. DIEHL ..... Professeur honoraire
M. HAMANT ..... Professeur honoraire
Mlle BERDUCOU .... Professeur honoraire

PROFESSEURS DE CLASSE EXCEPTIONNELLE ET DE 1ère CLASSE

M. NOUGARO ..... Hydraulique Générale et Appliquée
M. GARDY ..... Génie Chimique
M. VOIGT ..... Chimie Minérale
M. THIRRIOT ..... Hydraulique
M. GRUAT ..... Hydraulique
M. BUGAREL ..... Génie Chimique
M. DAT ..... Hydraulique
M. MARTY ..... Electronique
M. ANGELINO ..... Génie Chimique
M. HOFFMANN ..... Electronique
M. TRANNOY ..... Electrotechnique
M. FALLOT ..... Biotechnologie Végétale Appliquée
M. DABOSI ..... Métallurgie et Réfractaires
M. ENJALBERT .... Génie Chimique

PROFESSEURS DE 2ème CLASSE

M. TRUCHASSON ... Hydraulique
M. LEFEUVRE ..... Electronique
M. CALMON ..... Chimie Agricole
M. GILOT ..... Génie Chimique
M. MATHIEU ..... Chimie Appliquée
M. BAUDRAND ..... Electronique
M. BOURGEAT ..... Pédologie
M. MATHEAU ..... Electronique
M. BAJON ..... Electronique
M. COUDERC ..... Génie Chimique
M. GOURDENNE ... Chimie
M. LENZI ..... Chimie Industrielle
M. MASBERNAT .... Hydraulique
M. TERRON ..... Zoologie
M. BUIS ..... Biologie Quantitative
M. CONSTANT ..... Chimie Minérale
M. COSTES ..... Electrotechnique
M. ECOCHARD ..... Agronomie
M. CANDAU ..... Zootechnie
M. LABAT ..... Ichtyologie Appliquée
M. MORELIERE .... Electronique
M. GASET ..... Chimie Industrielle
M. BRUEL ..... Informatique
M. ALBERTINI ..... Cytologie et Pathologie Végétales
M. BELLET ..... Mécanique - Hydraulique
M. FABRE ..... Mécanique - Hydraulique
M. FOCH ..... Electronique, Electrotechnique, Automatique
M. MORA ..... Génie Chimique
M. MORARD ..... Physiologie Végétale Appliquée
M. NOAILLES ..... Mathématiques
M. BONEL ..... Chimie Appliquée
M. KALCK ..... Chimie Minérale
M. LAGUERIE ..... Génie Chimique
M. PECH ..... Sciences Agronomiques
M. PLANCHON ..... Sciences Agronomiques
M. GIBERT ..... Génie Chimique
M. ANDRE ..... Sciences Agronomiques
M. BEN AIM ..... Génie Chimique
M. BOUCHER ..... Electrotechnique, Electronique
M. CASTANIE ..... Automatiques, Informatique Industrielle
M. De FORNEL ..... Electrotechnique, Electronique
M. DOMENECH ..... Dynamique et Optimisation des procédés
M. FARRENY ..... Informatique Fondamentale Appliquée
M. HA MINH ..... Mécanique
M. MOLINIER ..... Génie Chimique
M. MONCOULON ... Sciences Agronomiques
M. PAREILLEUX ... Sciences Agronomiques
M. RIBA ..... Génie Chimique
M. ROBERT ..... Génie Chimique
M. RODRIGUEZ .... Systèmes
M. SALLE ..... Informatique Fondamentale Appliquée

ADMINISTRATION

M. CRAMPES ..... Secrétaire Général

INSTITUT DES SCIENCES APPLIQUEES DE TOULOUSE

M. SARAZIN ..... Directeur Général

PROFESSEURS DE CLASSE EXCEPTIONNELLE ET DE 1ère CLASSE

M. ROQUES ..... Chimie Appliquée, Génie Chimique
M. GRATELOUP .... Electronique, Electrotechnique, Automatique
M. MASO ..... Génie Civil
M. MIRA ..... Electronique, Electrotechnique, Automatique
M. DURAND Gilbert . Biologie et Biochimie Appliquée
M. ASKENAZY ..... Physique Atomique, Physique du Solide
M. LETURCQ ..... Electronique, Electrotechnique, Automatique
M. SCHUTTLER .... Electronique, Electrotechnique, Automatique
M. TITLI ..... Electronique, Electrotechnique, Automatique
M. BROUSSEAU .... Physique Atomique, Physique du Solide

PROFESSEURS DE 2ème CLASSE

M. FAGET ..... Physique Atomique, Physique du Solide
M. BARTHET ..... Mécanique
M. SIRIEYS ..... Mécanique
M. BESOMBES-VAILHE ..... Chimie Appliquée, Génie Chimique
M. VERDIER ..... Chimie Organique, Minérale et Analytique
M. BOUDET René ... Génie Mécanique
M. JAVELAS ..... Génie Civil
M. LACAZE ..... Mathématiques II
M. LORRAIN ..... Génie Civil
M. MARTINEZ ..... Electronique, Electrotechnique, Automatique
M. GOMA ..... Biologie et Biochimie Appliquées
M. LAFFITTE ..... Génie Civil
M. PORTAL ..... Physique Atomique, Physique du Solide
M. MONSAN ..... Biologie et Biochimie Appliquée
M. FANTIN ..... Automatique et Informatique Industrielle
M. OLLIVIER ..... Génie Civil

ADMINISTRATION

M. CROS ..... Secrétaire Général

REMERCIEMENTS

Je tiens à exprimer ma reconnaissance et mes remerciements à Messieurs :

Georges GIRALT, Directeur de Recherche au C.N.R.S.,

Marc COURVOISIER, Professeur à l'Université Paul Sabatier, Toulouse,

Henri FARRENY, Professeur à l'I.N.P. de Toulouse,

Hervé GALLAIRE, Responsable du Groupe Informatique aux Laboratoires de Marcoussis, Centre de Recherche de la C.G.E.,

Gérard GUIHO, Professeur à l'Université Paris-Sud, Orsay,

Richard M. KARP, Professeur à l'Université de Californie, Berkeley,

Maurice NIVAT, Professeur à l'Université de Paris VII, et

Roland PRAJOUX, Maître de Recherche au C.N.R.S.,

pour avoir bien voulu critiquer et évaluer ce travail et me faire l'honneur de participer à mon jury de thèse.

Je suis, par ailleurs, reconnaissant :

- à Roland PRAJOUX pour l'amitié qu'il m'a toujours témoignée,
- à Dick KARP pour l'accueil scientifique qu'il m'a réservé durant mon séjour très fructueux à Berkeley, et pour m'avoir initié à l'algorithmique et sa complexité,
- à Hervé GALLAIRE pour son enseignement et ses conseils que j'ai constamment appréciés, et pour l'intérêt bienveillant qu'il a continué à porter depuis de longues années au travail d'un de ses anciens élèves,
- à Daniel ESTEVE, Directeur du Laboratoire d'Automatique et d'Analyse des Systèmes du C.N.R.S., pour le cadre de recherche idéal qu'est le L.A.A.S., où je suis resté depuis qu'il m'a accueilli en tant qu'élève-ingénieur, stagiaire pour 3 mois dans le Groupe "Automatisation Intégrée",

et particulièrement :

- à Georges GIRALT, pour la confiance qu'il a bien voulu m'accorder, et pour sa direction et son encadrement attentif de ce travail depuis ses premiers débuts. J'ai été très sensible à ses encouragements et à l'intérêt particulier qu'il a porté à mon sujet de recherche. Au delà de l'environnement et du contexte scientifique très riche dont il m'a fait bénéficier dans son équipe, ce mémoire lui est redevable de nombreuses suggestions et idées fructueuses.

Mes remerciements vont également :

- à tous mes camarades de l'Equipe de Robotique du L.A.A.S. pour les discussions et échanges nombreux et féconds que j'ai pu avoir avec eux, et pour leur collaboration et aide scientifique très appréciable, dont plusieurs sections de cette thèse ont profité,
- à tous mes collègues du L.A.A.S. et d'autres laboratoires toulousains, en particulier ceux du L.S.I. (Equipe d'Intelligence Artificielle), qui m'ont aidé à animer le séminaire de Robotique,
- à Melle H. CAMARASA, Mme J. PENAVAYRE et à E. LAPEYRE-MESTRE et ses collaborateurs du Service Documentation et Tirage pour leurs interventions compétentes dans la réalisation de ce mémoire.

Enfin, je terminerai en témoignant ma gratitude à ma famille et à tous mes proches pour leur patience et leur compréhension.

°0°

TABLE DES MATIERES

INTRODUCTION 1

1ÈRE PARTIE : MODÈLES ET ALGORITHMES

CHAPITRE 1. MODÈLES ET FORMALISMES DE PROCESSUS DÉCISIONNELS FERMÉS 17

I.1. INTRODUCTION 19

I.2. MODELES DE BASE 20

I.2.1. Arbres de décision et partitions sur un hypercube 20
I.2.2. Procédures d'identification et questionnaires 28
I.2.3. Tables de décision 32

I.3. FORMULATION D'UN MODELE GENERAL ET PRINCIPALES PROPRIETES 37

I.3.1. Systèmes de règles de décision 37
I.3.2. Consistance et complétude d'un système de règles 40
I.3.3. Processus décisionnel pour un système de règles 44

I.4. PARAMETRES ET CRITERES D'OPTIMISATION 48

I.4.1. Distributions de probabilité 48
I.4.2. Modèle des coûts unitaires 50
I.4.3. Modèle des coûts constants 50
I.4.4. Modèle des coûts dépendants 52
I.4.5. Modèle des coûts conditionnels 53
I.4.6. Modèle des coûts variables 53
I.4.7. Modèle des coûts d'accès 54
I.4.8. Modèle des coûts de maintenance 55

I.5. ALGORITHMES APPROCHES D'OPTIMISATION 58

I.5.1. Un algorithme non déterministe 58
I.5.2. Diverses fonctions de choix heuristiques 62

I.6. ALGORITHMES EXACTS D'OPTIMISATION 67

I.6.1. Programmation Dynamique 67

I.6.2. Procédure de séparation et d'évaluation - Recherche arborescente 70

I.7. CONCLUSION 77

CHAPITRE 2. ALGORITHMES QUASI-ADMISSIBLES DE RECHERCHE HEURISTIQUE DANS LES GRAPHES ET LES HYPERGRAPHES 79

2.1. INTRODUCTION 81

2.2. RECHERCHE HEURISTIQUE ADMISSIBLE DANS UN GRAPHE : A^* 84

Convergence et admissibilité d'A^*
Complexité d'A^*
Monotonie de l'heuristique

2.3. RECHERCHE HEURISTIQUE ϵ-ADMISSIBLE DANS UN GRAPHE : A_ϵ 98

Heuristique monotone
Heuristique α-minorante
Complexité d'A_ϵ

2.4. RECHERCHE HEURISTIQUE DANS UN HYPERGRAPHE : AO^* 114

2.5. RECHERCHE HEURISTIQUE DANS UN HYPERGRAPHE : AO_ϵ 127

Complexité de la recherche heuristique dans un hypergraphe
Stratégie persévérante pour AO_ϵ
Généralisation à des hypergraphes orientés quelconques

2.6. RECHERCHE HEURISTIQUE ET OPTIMISATION COMBINATOIRE 153

2.7. CONCLUSION 160

CHAPITRE 3. SCHÉMA D'APPROXIMATION POUR PROCESSUS DÉCISIONNELS FERMÉS 161

3.1. INTRODUCTION 163

3.2. UN SCHEMA D'APPROXIMATION POUR PROCESSUS DECISIONNELS FERMES 164

3.2.1. Un espace de recherche 164
3.2.2. Un estimateur heuristique 168
3.2.3. Un schéma d'approximation 176
3.2.4. Post-optimisation : améliorations d'un arbre de décision 187

3.3. GENERALISATIONS PARAMETRIQUES 192

3.3.1. Modèles des coûts conditionnels et des coûts dépendants 192
3.3.2. Modèles des coûts variables 194
3.3.3. Modèles des coûts d'accès 197
3.3.4. Modèles des coûts de maintenance 208

3.4. GENERALISATIONS STRUCTURELLES 210

3.4.1. Restriction du domaine des caractéristiques et contraintes de succession 215
3.4.2. Règle Autre 217
3.4.3. Systèmes de règles récursifs interconnectés 217

3.5. CONCLUSION 221

CHAPITRE 4. COMPLEXITÉ DE L'OPTIMISATION DES PROCESSUS DÉCISIONNELS FERMÉS 224

4.1. INTRODUCTION 225

4.2. CARACTERISATION DE LA COMPLEXITE DU PROBLEME D'OPTIMISATION DES PDF 227

4.2.1. Dimension d'un système de règles 227
4.2.2. Le problème d'optimisation, exacte ou approchée 229
4.2.3. Le problème d'existence correspondant 238

4.3. MODELE EXPERIMENTAL DE LA COMPLEXITE MOYENNE D'AO$_\epsilon$ 244

4.3.1. Implémentation 246

4.3.2. Expérimentation 252

4.3.3. Modèle et test de stratégies d'exploration 255

4.4. APPROCHE POUR UNE OPTIMISATION DYNAMIQUE 271

4.4.1. Domaine d'application de l'algorithme 272

4.4.2. Evolutions paramétriques 275

4.4.3. Evolutions structurales 279

4.5. CONCLUSION 282

2ÈME PARTIE : QUELQUES PROCESSUS DÉCISIONNELS EN ROBOTIQUE

CHAPITRE 5. UN SYSTÈME D'IDENTIFICATION D'OBJETS 285

5.1. INTRODUCTION : PRESENTATION DE L'APPLICATION 287

5.2. IDENTIFICATION D'OBJETS COMPLETEMENT OBSERVES 294

5.2.1. La Reconnaissance de Formes Séquentielle : approches connues et leurs difficultés 294

5.2.2. Partitionnement de l'espace de représentation 301

5.2.3. Génération d'un système de règles d'identification 310

5.2.4. Génération du classifieur 314

5.2.5. Mise en oeuvre expérimentale 319

5.3. APPROCHE A UNE IDENTIFICATION SYNTAXIQUE D'OBJETS PARTIELLEMENT OBSERVES 321

5.3.1. Modèle et représentation d'objets plans 321

5.3.2. Apprentissage 326

5.3.3. Identification 328

5.3.4. Algorithmes de recherche de facteurs dans un mot 330

5.4. CONCLUSION 349

CHAPITRE 6. PROCESSUS DÉCISIONNELS OUVERTS POUR LA GÉNÉRATION DE PLANS EN ROBOTIQUE 351

6.1. INTRODUCTION 353

6.2. MODELES DE PROCESSUS DECISIONNELS OUVERTS 355

6.2.1. Formalisme des Systèmes de Règles de Production 356

6.2.2. Formalisme de la Logique des Prédicats du premier Ordre 361

6.3. L'UNIFICATION PAR UN PROCESSUS DECISIONNEL FERME 365

6.3.1. Position du problème 365

6.3.2. Approches communes 368

6.3.3. Arbre d'unification 372

6.3.4. Quelques extensions 381

6.4. INSTANCIATION DES REGLES 390

6.4.1. Instanciation exhaustive 390

6.4.2. Instanciation sélective 396

6.5. APPROCHE A UN PDO GUIDE PAR RECHERCHE HEURISTIQUE 399

6.6. CONCLUSION 402

CONCLUSION 403

ANNEXE 409

BIBLIOGRAPHIE 419

INTRODUCTION

Dans ce mémoire consacré à l'"Optimisation de Processus Décisionnels pour la Robotique", je présente, dans une problématique unifiée, des contributions sur les thèmes clés de :

(i) Processus Décisionnels,

(ii) Optimisation combinatoire, et

(iii) Application à la Robotique.

L'introduction de ces thèmes et leur adéquation dans la problématique adoptée sont développés dans ce qui suit.

(i) Processus Décisionnels

Le terme "décision" figure dans de nombreuses disciplines scientifiques avec des sémantiques très diverses (par exemple théorie de la décision [246], analyse de la décision [199], problèmes décisionnels en inférence logique [120], ...). Il semble curieux que celle initialement rattachée à l'expression "Processus Décisionnels" fasse uniquement référence au contexte de l'optimisation combinatoire, et en particulier à la programmation dynamique considérée comme procédure d'optimisation par une séquence de choix (de décisions). Les premières formulations dans ce sens seraient dues à DENARDO [30] puis KARP et HELD [115] qui définissent les notions de Processus Décisionnels Discret et de Processus Décisionnels Séquentiels pour modéliser la Programmation Dynamique. IBARAKI [101] apporte des généralisations sur ces modèles (Processus Décisionnels Séquentiels Monotones), y incluant le "Branch-and-Bound".

On retiendra ici les caractéristiques fondamentales de processus discret et séquentiel, mais on adoptera une définition, plus générale que celles citées, et qui permet de retrouver les procédures d'optimisation combinatoires comme cas particulier intéressant.

On part initialement d'un formalisme qui intervient fréquemment en résolution automatique de problèmes, dans de nombreuses autres applications de l'Intelligence Artificielle, et qui est fondamental dans tous les aspects décisionnels en robotique : celui de système de régles de décision.

Un tel système est représenté dans un espace d'état discret (fini ou infini), et on dispose d'un ensemble (fini) de règles qui appliquent l'espace d'état sur un ensemble d'actions, définies au préalable, ces actions entrainant éventuellement un changement d'état du système. Chaque règle est du type : "Si (Antécédent) alors (Conséquent)" ; l'antécédent caractérise l'état, ou le sous-ensemble d'états, du système auxquels doivent être associées l'action, ou la suite d'actions décrites dans le conséquent.

Sur ce modèle de base, on distinguera deux types de processus décisionnels : les processus Fermés (PDF) et les processus Ouverts (PDO).

Un PDF consiste en une suite d'opérations d'acquisitions et de traitements d'informations qui permettent de caractériser (partiellement) l'état du système à un moment donné, pour déterminer la règle de décision qui lui correspond. La mise en oeuvre des actions associées à cette règle, et le changement d'état éventuel qui en résulte ne sont pas du ressort direct du PDF.

Un PDO a pour but, au contraire, de déterminer une séquence de règles qui amènerait le système d'un état initial à un état objectif.

Alors que dans un PDF à un état donné correspond une et une seule règle de décision, dans un PDO l'ensemble des règles ne définissent qu'une application multivoque ; un des problèmes étant alors de choisir une règle particulière parmi celles associées à un état, et de revenir éventuellement sur ce choix.

Par ailleurs, on peut toujours se ramener dans un PDF à un espace de représentation fini. L'espace d'état dans un PDO sera en général infini.

Finalement, aussi bien dans un PDF que dans un PDO on décrira les antécédents des règles par des expressions logiques. L'état du système sera alors un modèle d'interprétation pour ces expressions : une règle est associée aux états pour lesquels son antécédent est valide. Mais dans un PDF, on se contentera d'expressions de logique propositionnelle (ne contenant que la seule variable d'état), alors que dans un PDO, les antécédents des règles seront exprimés en logique des prédicats du 1er ordre, avec des variables quantifiées.

Ces deux types de processus décisionnels apparaissent dans la littérature dans des domaines complétement indépendants et cloisonnés. On trouvera des modèles simplifiés de PDF dans les articles traitant de Procédures d'Identification, Questionnaires, et Tables de Décision. Les PDO quant à eux figurent dans le domaine de l'Intelligence Artificielle et en particulier des Systèmes de Règles de Production. Malgré les différences entre PDF et PDO qui viennent d'être soulignées, il nous a paru fructueux de rapprocher ces deux types de processus, entre autres pour les raisons suivantes :

- Un PDF est un cas particulier simple de PDO pour lequel on dispose de nombreux outils efficaces de traitement. Il est donc important de reconnaître le cas particulier "fermé" à chaque fois qu'il se présente. Or ceci est très rarement fait en Intelligence Artificielle (je ne connais qu'une seule exception : le système de diagnostic médical IRIS [234] organisé en un réseau de tables de décision). Au contraire, de nombreux "Systèmes Experts" ne faisant intervenir que des PDF sont présentés dans le formalisme général des Règles de Production (PDO). Ainsi les exemples de ce formalisme donnés dans [245] sont immédiatement exprimables en tables de décision. Un cas plus probant est celui du système Prospector [35], qui, indépendamment de son mécanisme d'actualisation des probabilités a postériori [183], ne fait qu'intervenir à mon avis qu'un PDF (cf. la compilation de ses règles en arbre de décision [127]).

- Il apparaît intuitivement de la présentation rapide qui vient d'être faite qu'un PDO peut être organisé en une séquence de PDF. On verra dans le Chapitre 6 l'avantage qui peut en être tiré.

- Finalement, le rapprochement PDF/PDO sera prolongé au niveau des algorithmes d'optimisation des procédures qui y interviennent. Le même algorithme sera utilisé dans les deux cas : ce qui est outil d'optimisation dans un PDF devient l'objet à optimiser dans un PDO ; et ce qui est complexité d'un algorithme dans le 1er cas, devient coût d'un processus dans le second.

Remarques :

- La terminologie Processus Décisionnels Fermés/Ouverts qui est adoptée fait référence aux aspects : espace d'état fini/infini, ou règle unique/séquence de règles ; on peut aussi la rapprocher de : clause fermée (sans variable quantifiée)/clause ouverte.

- Une analogie pour le lecteur automaticien : un PDF serait à mettre en parallèle avec la notion d'observabilité, alors qu'un PDO le serait avec celle de gouvernabilité (en notant la différence entre l'existence d'une propriété et le processus correspondant).

(ii) Optimisation combinatoire

On s'intéresse dans ce mémoire à la modélisation des Processus Décisionnels, à leur analyse, et surtout à leur optimisation.

Un Processus Décisionnel est considéré comme une procédure particulière, et l'optimisation porte sur sa complexité, selon différents modèles et critères de coût. La prise en compte dans ces critères de la complexité de la tâche d'optimisation elle-même, et la caractérisation de cette tâche en tant que problème NP-dur, conduisent à défendre la thèse de l' ϵ-optimisation par des algorithmes généraux à approximation garantie (schémas d'approximation). Montrons rapidement pourquoi.

Il est aujourd'hui largement reconnu que l'optimisation exacte d'un problème NP-dur ne pourra se faire que par un algorithme énumératif et n'échappera pas à une complexité exponentielle. Il est par contre beaucoup moins admis que pour de très nombreux problèmes, même une optimisation approchée relèvera du même sort. On pense le plus souvent qu'en exploitant la structure particulière d'un problème difficile, on pourra trouver pour le résoudre un algorithme polynomial fournissant une bonne approximation (de préférence quantifiée et bornée d'une façon ou d'une autre).

Il est exact que plusieurs problèmes se sont prêtés à cette approche (cf. par exemple [106]), et des techniques pour la construction de schéma d'approximation ont été dégagées [217]. Mais GAREY et JOHNSON ont pu montrer qu'elles ne sont en fait applicables qu'à des problèmes dont le caractère NP-dur est dû uniquement à l'amplitude des paramètres qui y interviennent, et qu'elles échouent sur tous les problèmes NP-durs au sens fort [62].

Quelques problèmes relevant de cette dernière catégorie ont été cependant résolus avec de bonnes approximations (constantes) par des algorithmes polynoniaux du type "glouton" (cf. exemples dans [60]). Cependant, même pour ce type d'algorithmes des résultats négatifs ont récemment été établis sur une large classe de problèmes. Analysant la minimisation de "Systèmes de dépendance" (structures plus générales que les matroïdes, définies par : famille S d'éléments de $\mathcal{P}(E)$ pour E ensemble fini, telle que $\forall s \in S : s' \subset s \Rightarrow s' \in S$) pour des critères addifits, KORTE [128] a établi que l'algorithme "glouton" pouvait fournir des solutions arbitrairement éloignées du minimum, et que de plus il n'existe aucun algorithme d'approximation meilleure que l'algorithme "glouton" (en proximité à l'optimum) tout en étant polynomial.

Ces résultats éclairent les constatations faites (section 1.5.2.) à propos de l'utilisation de l'approche "glouton" pour notre problème d'optimisation des PDF ; et justifient le recours à un schéma d'approximation (écart à l'optimum garanti). De plus, compte tenu de la caractérisation (Chapitre 4) de ce problème comme étant à approximation NP-dur , un tel schéma ne peut reposer que sur un algorithme général du type semi-énumératif. Finalement, l'approche "schéma d'approximation" permet de paramétrer l'optimisation et d'envisager l'inclusion de la complexité de l'optimisation dans le critère global à optimiser.

Pour résumer, la problématique d'optimisation adoptée dans ce mémoire est celle de la complexité :

a) optimiser la complexité de PDF selon divers modèles,
b) analyser et modéliser la complexité des algorithmes d'optimisation utilisés en (a), et
c) considérer un PDO comme équivalent à une procédure d'optimisation de PDF, et minimiser sa complexité grâce aux modèles de (b) (point envisagé à un niveau prospectif).

(iii) Applications à la Robotique

La robotique est essentiellement multidisciplinaire, et ce n'est pas faute de les ignorer qu'on ne mentionnera même pas dans ce mémoire un grand nombre (sinon la majorité) des thèmes et des problèmes importants qui s'y posent (cf. [74] pour une présentation générale).

L'ambition de ce travail est d'amener à la robotique une contribution extrêmement limitée, dans le cadre étroit de notre problématique d'optimisation des processus décisionnels.

S'il semble inutile de justifier l'existence de problèmes décisionnels dans la robotique d'aujourd'hui (malgré les réserves qui s'expriment parfois sur ce point), il est par contre nécessaire d'argumenter en faveur de la nécessité d'une optimisation. Pourquoi, demanderait-on, s'intéresser à l'optimisation de processus décisionnels dans un domaine où même la définition de tels processus est encore problématique ?

Le principal argument de réponse, à mon avis, est que cette même définition se pose le plus souvent en termes d'optimisation. A la base de tous les processus décisionnels en robotique (du moins, ceux à ma connaissance) se trouve le problème d'un ensemble de choix sur un univers fortement combinatoire ; et la définition du processus serait triviale si on pouvait résoudre ce problème d'une manière exhaustive. Comme dans tous les cas concrets, on ne le peut pas, la nécessité d'une sélection fait intervenir (éventuellement à plusieurs niveaux) une optimisation, au sens large.

Illustrons ceci par quelques exemples. En classification - identification d'objets, on peut définir a priori un nombre très grand de caractéristiques redondantes, et la sélection de celles pertinentes pour une application particulière se résoud par l'optimisation du classifieur. En génération de plans, non seulement il n'est pas question d'essayer toutes les séquences d'opérateurs d'une certaine longueur, mais de plus, même l'application d'une procédure d'inférence, telle que le Principe de Résolution, est impraticable sans de puissantes heuristiques, de bons algorithmes et stratégies pour guider l'inférence, et même une optimisation des opérations élémentaires qui y interviennent le plus fréquemment. On peut ajouter d'autres exemples, notamment ceux faisant intervenir l'apprentissage.

Un deuxième argument peut être invoqué en faveur de l'optimisation : dans le contexte de la robotique, tout problème est par nature fortement contraint. Cela est vrai même pour les problèmes décisionnels : coûts économiques de mise en oeuvre d'un processus décisionnel, temps de réponse de ce processus, coûts de la solution fournie, éventuellement sa fiabilité, etc.

Je n'essayerai pas davantage de convaincre le lecteur sceptique qu'une approche telle que celle exposée peut amener une contribution effective au domaine de la robotique. Je tenterai par contre de montrer ce que la robotique a amené (en contre-partie ?) à cette problématique.

Sur un plan général, j'ai bénéficié d'un support de réflexion extrêmement riche. Comme il a été dit, on rencontre en robotique un grand nombre de problèmes de nature combinatoire, et on peut trouver en chacun de multiples extensions et ramifications intéressantes (par exemple, tous les modèles de coûts d'optimisation abordés ici ont été suggérés par des applications concrètes). De plus, ces problèmes se posent d'une manière très précise, ou bien leur formulation peut être précisée par confrontation à l'application robotique (ce qui est particulièrement important pour les problèmes du type Intelligence Artificielle).

D'une manière plus spécifique, une des principales raisons pour laquelle ce mémoire défend une thèse sur l'approximation algorithmique provient de son développement dans le contexte d'applications à la robotique. Si l'intégration des multiples contraintes d'une de ces applications rend l'optimisation nécessaire, elle exclut une optimisation stricte, par rapport à un seul critère et sur un seul sous-système. Le souci d'intégration conduit au contraire à prendre en compte, dans une approche globale, toutes les ressources du robot, y compris ses ressources de calcul .

Les coûts associés à ces calculs sont rarement négligeables, et souvent du même ordre de grandeur que ceux correspondants au critère à optimiser. La relation est critique dans le cas de la génération d'un plan exécuté une seule fois (un robot qui, pour gagner 1 ou 2 minutes dans son chemin entre 2 points, consacrerait 15 à 20 minutes à optimiser sa trajectoire aura un sens bien curieux de l'économie).

De ces considérations découlent, et la nécessité d'algorithmes fournissant une approximation quantifiée et paramétrable, et celle de bons modèles du comportement de ces algorithmes.

Pour éclairer la démarche suivie et montrer l'adéquation d'une problématique scientifique (en définitive très appliquée) avec son domaine d'application, cette rapide présentation a développé des considérations prospectives et mentionné des objectifs non encore atteints. Insistons pour que cette introduction n'induise pas en erreur : le mémoire présenté ici ne se propose d'amener que quelques contributions limitées dans le sens de ces objectifs.

Le traitement de la problématique qui vient d'être exposée s'articulera autour de 3 thèmes :

1. Outils : développement d'algorithmes de recherche heuristique admissibles et ϵ-admissibles dans les graphes et les hypergraphes, considérés comme des procédures générales de recherche et d'optimisation (schémas d'approximation) pour une large classe de problèmes discrets.

2. Concepts : modélisation, analyse et optimisation, par les outils précédents, de Processus Décisionnels ; la contribution portant essentiellement sur les PDF, étudiés directement ou en tant qu'éléments constituants des PDO ; et en présentant quelques éléments prospectifs sur la modélisation et l'optimisation des PDO par les méthodes de recherche heuristique.

3. Applications : étude de deux Processus Décisionnels en Robotique, l'un fermé portant sur un problème d'apprentissage de classifieur pour l'identification d'objets, et l'autre ouvert traitant de génération de plans.

Le premier thème fait l'objet du Chapitre 2 (qui peut être abordé indépendamment du reste du mémoire). On y propose 2 algorithmes originaux : A_ϵ pour la recherche heuristique dans les graphes, et AO_ϵ pour celle dans les hypergraphes. Chacun de ces algorithmes peut être considéré comme une classe de procédures paramétrées dans laquelle l'utilisateur, en fixant une valeur du paramètre ϵ et en optant pour certaines stratégies, choisit

un algorithme particulier réalisant un compromis entre les deux critères : coût de la solution requise - coût de sa recherche. Ces classes comprennent comme cas particuliers les algorithmes A^* et AO^* . Le Chapitre reprend, avec quelques améliorations, une présentation synthétique de ces deux derniers algorithmes, en développant de nouvelles preuves de propriétés connues et quelques propriétés originales, ainsi qu'une étude de leur complexité. Ceci conduit à introduire et à justifier les généralisations proposées : A_ϵ et AO_ϵ .

On démontre les théorèmes de convergence et d' ϵ-admissibilité, et on analyse les comportements de ces algorithmes, pour plusieurs types de fonctions heuristiques et de stratégies d'exploration, et différentes structures de l'espace de recherche. Finalement, on propose une jonction originale entre les méthodes d'optimisation en Recherche Opérationnelle et celles de résolution de problèmes en Intelligence Artificielle : on compare les algorithmiques du type A^* aux procédures de Séparation et d'Evaluation Progressives (Branch-and-Bound) et on démontre l'inclusion de ces dernières comme cas particulier des premiers. Les contributions de ce Chapitre ont été partiellement publiées dans [71-72].

Le deuxième thème est traité plus longuement en trois chapitres (1, 3 et 4).

Le Chapitre 1 est, pour la plupart de ses sections, un Chapitre de synthèse. Il introduit un modèle général de PDF à partir des modèles connus : Questionnaires et Tables de Décision. Les outils de bases, arbres de décision, partitions sur le n-cube, sont exposés. Le problème d'optimisation est ensuite abordé, en proposant 5 modèles de coût originaux, pour la plupart compatibles additivement entre-eux, et dont la nécessité s'impose à partir de la formulation d'un PDF en algorithme et de sa complexité en coût.

Le Chapitre 3 applique l'algorithme AO_ϵ à cette optimisation. On en propose une instance exploitant efficacement la structure particulière du problème, et on analyse de nombreuses variantes de stratégies d'exploration ainsi que les phases de la recherche où elles devraient intervenir.

Pour chacun des modèles de coût, on développe une fonction d'évaluation heuristique dont on démontre les propriétés assurant l' ϵ-admissibilité de l'algorithme. Le modèle des coûts d'accès est particulièrement intéressant : l'évaluation de l'heuristique fait intervenir une deuxième optimisation combinatoire par résolution approchée du problème dual du "knapsack". On s'intéresse finalement à certaines structures particulières de PDF et on propose des heuristiques pour les traiter.

Le Chapitre 4 caractérise théoriquement la complexité du problème d'optimisation, exacte ou approchée, des PDF. Par plusieurs réductions polynomiales, on établit les théorèmes d'appartenance à la classe des problèmes NP-dur au sens fort, de la non-appartenance du problème décisionnel correspondant à la classe NP, et, même en se restreignant aux instances où ce problème est dans NP, de l'inexistance d'algorithmes d'optimisation exacte polynomiaux, de schéma d'approximation pleinement polynomiaux ou simplement polynomiaux, ou d'algorithmes à approximation garantie absolue ou relative polynomiaux. On développe ensuite un modèle empirique du comportement moyen de l'algorithme AO_ϵ dans le cas de l'optimisation des PDF, d'où il ressort une complexité variant entre $O(2^n)$ et $O(1.3^n)$ pour ϵ variant de 0 à l'infini. Le Chapitre se termine par une section prospective : si on s'intéresse à un système de règles dont les paramètres et éventuellement même la structure évoluent dynamiquement, conduisant à n'utiliser le même PDF qu'un nombre limité de fois, et si l'on dispose d'un modèle fiable de la complexité (traduite en coût) de l'optimisation des PDF, on peut alors se poser les problèmes suivants : quel est le domaine précis d'application de l'algorithme ?, et si on se trouve dans le domaine où une optimisation se justifie, quel degré d'approximation adopter une première fois ?, et comment suivre la dynamique du système pour remettre en cause la solution utilisée et en rechercher une autre meilleure ? On propose quelques contributions dans cette direction .

Cetains des résultats développés dans les Chapitres 3 et 4 ont été décrits dans [67-68].

Le dernier thème abordé, celui des applications à la robotique fait l'objet d'une deuxième partie du mémoire comprenant les Chapitres 5 et 6.

On consacre le Chapitre 5 aux méthodes décisionnelles en identification d'objets représentés par des images à 2 dimensions. L'application est située dans le contexte de la robotique, avec les contraintes de coût, fiabilité et temps de réponse qui lui sont spécifiques, et en justifiant la nécessité d'un apprentissage automatique et d'une optimisation du classifieur. Une première section traite d'identification d'objets totalement observés, utilisant une Reconnaissance de Formes Séquentielles (RFS) et une représentation par caractéristiques globales. Après une présentation critique des principales techniques de RFS connues, on propose une méthode originale en trois phases : 1) partitionnement de l'espace de représentation, 2) synthèse d'un système de règles d'identification, et 3) génération d'un classifieur en PDF arborescent minimal pour le coût moyen d'identification et ayant un taux d'erreur inférieur à une contrainte spécifiée a priori, et un taux de non classification faible. Une implémentation est décrite. La section suivante du Chapitre aborde le problème d'identification d'objets partiellement observés : on expose la problématique et quelques approches de résolution du problème d'apprentissage par des techniques de recherche de séquences (locales) caractéristiques dans une représentation de contours codée sur un alphabet ; ce qui fait intervenir le problème de recherche de facteurs dans un mot (string matching). Le Chapitre se termine par une contribution algorithmique à ce problème qui se pose dans de nombreuses applications. L'algorithme proposé généralise les deux principaux algorithmes connus, basés sur la comparaison dans l'ordre gauche-droite (Knuth-Morris-Pratt [124]) ou droite-gauche (Boyer-Moore [18]). Son originalité est de déterminer, pour un facteur donné, un ordre de comparaison optimal en moyenne sur tout texte dont on connaîtrait la distribution des lettres. On démontre la validité de l'algorithme, on décrit une implémentation, et on analyse son comportement dans les phases de prétraitement et de recherche, en le comparant à d'autres algorithmes. Quelques extensions à des problèmes dérivés sont étudiées.

Les contributions proposées en 1ère partie de ce Chapitre ont fait l'objet des publications suivantes [73-75-76].

Le Chapitre 6 traite de la génération de plan en robotique, dans la problématique de la complexité des processus décisionnels qui y interviennent. On introduit les systèmes de Règles de Production comme modèle de PDO, puis on aborde le problème de l'unification (Pattern-matching) dont on justifie l'importance cruciale pour la complexité de ces processus. On en discute les alternatives (unification/ demi-unification, 1er ordre/2ème ordre) en complexité du PDO par rapport à la puissance expressive des règles ; et on argumente en faveur d'une demi-unification avec usage restreint des opérateurs du type "segment". Ce cas est traité en considérant le problème : "à quels prédicats s'unifie telle donnée ?" comme en PDF ; on lui applique les outils et concepts développés en 1ère partie du mémoire. Quelques exemples obtenus sur une implémentation en cours sont exposés. On propose ensuite des procédures d'instanciation des PDO en considérant les 2 cas : recherche de toutes les instances valides, et recherche d'une ou de quelques instances particulières. La dernière section développe une approche à l'application des algorithmes de recherche heuristique tels que A_ϵ ou AO_ϵ pour guider et optimiser un PDO. Pour l'essentiel, les contributions de ce Chapitre ont été décrites dans [69-70].

Le plan de ce mémoire n'a pas échappé à la difficulté de la projection d'une structure thématique, où les adjacences sont multiples, en une séquence linéaire. Malgré un souci d'unité et de cohésion, nous avons tenté de simplifier la lecture en modularisant certaines sections.

Le Chapitre 2 peut être lu de façon autonome. De plus chacune de ses parties, recherche dans un graphe (2.2. et 2.3.) et recherche dans un hypergraphe (2.4. et 2.5.), est dans une certaine mesure indépendante de l'autre.

Dans une première lecture, les deux Chapitres 5 et 6 d'applications à la robotique peuvent également être abordé séparemment l'un de l'autre et du reste du mémoire. La section sur le "string-matching" (5.3.4.) utilise les notations propres à ce domaine, et en dehors d'un algorithme d'optimisation combinatoire, fait peu référence au reste du mémoire.

Le lecteur familier des outils tables et arbres de décision, partitions sur le n-cube, peut restreindre sa lecture du Chapitre 1 de synthèse aux sections (1.3.),(1.4.) et 1.5.).

Le Chapitre 3 nécessite la lecture de la section sur AO_ϵ (2.5.) On peut cependant aborder chacun des modèles de coûts généralisant le modèle de base (3.3.) et chacune des généralisations structurales d'une façon indépendante des autres.

Enfin, la première partie du Chapitre 4 (section 4.2.) qui caractérise la complexité du problème d'optimisation des PDF ne fait référence à aucun algorithme spécifique et peut être lu directement après le Chapitre 1. Les deux autres parties (4.3. et 4.4.) nécessitent la lecture du début du Chapitre 3 (3.2.).

Nous avons essayé d'utiliser des notations homogènes tout le long du mémoire. Les changements de notations doivent être attribués (aux erreurs involontaires près) à une différence sémantique qui sera explicitée ultérieurement (par exemple u ; variable d'état dans le Chapitre sur la recherche heuristique correspond à un k-cube ou sous-ensemble d'états e du Chapitre 1, car u qualifie l'état de la recherche, et e celui du système sur lequel porte cette recherche).

Finalement, la contribution de cette thèse se situant dans le domaine algorithmique, nous avons essayé d'exposer formellement tous les algorithmes développés (en privilégiant cependant la lisibilité à une syntaxe lourde, et en utilisant la typographie et la numérotation pour les structurer), d'en proposer des preuves rigoureuses et des analyses de leur complexité. Le lecteur qui n'est pas particulièrement intéressé peut omettre ces preuves.

PREMIERE PARTIE

MODÈLES ET ALGORITHMES

CHAPITRE I

MODELES ET FORMALISMES DE PROCESSUS DECISIONNELS FERMES

I.1. INTRODUCTION

I.2. MODELES DE BASE

I.2.1. Arbres de décision et partitions sur un hypercube

I.2.2. Procédures d'identification et questionnaires

I.2.3. Tables de décision

I.3. FORMULATION D'UN MODELE GENERAL ET PRINCIPALES PROPRIETES

I.3.1. Systèmes de règles de décision

I.3.2. Consistance et complétude d'un système de règles

I.3.3. Processus décisionnel pour un système de règles

I.4. PARAMETRES ET CRITERES D'OPTIMISATION

I.4.1. Distributions de probabilité

I.4.2. Modèle des coûts unitaires

I.4.3. Modèle des coûts constants

I.4.4. Modèle des coûts dépendants

I.4.5. Modèle des coûts conditionnels

I.4.6. Modèle des coûts variables

I.4.7. Modèle des coûts d'accès

I.4.8. Modèle des coûts de maintenance

I.5. ALGORITHMES APPROCHES D'OPTIMISATION

I.5.1. Un algorithme non déterministe

I.5.2. Diverses fonctions de choix heuristiques

I.6. ALGORITHMES EXACTS D'OPTIMISATION

I.6.1. Programmation Dynamique

I.6.2. Procédure de séparation et d'évaluation - Recherche arborescente

I.7. CONCLUSION

I.1. INTRODUCTION

Un Processus Décisionnel Fermé (PDF) est une séquence d'acquisitions et de traitements d'informations sur un système dont l'espace de représentation est discret fini, conduisant à une décision de choix d'une action (ou d'une suite d'actions) dans un ensemble fini et connu. Ce choix repose sur un ensemble de règles de décision qui constitue une application de l'espace de représentation du système sur l'ensemble des suites d'actions possibles.

Ainsi informellement défini, un PDF est une généralisation de certains modèles du type procédure d'identification, questionnaire, ou table de décision. Les principales définitions et propriétés de ces diverses représentations sont rappelées dans la section suivante. La troisième section est consacrée au développement d'un modèle général reposant sur les deux notions de base de système de règles de décision, et procédure interprétant un arbre de décision représentant ce système. Certaines propriétés structurales de consistance et de complétude d'un système de règles sont présentées et un algorithme permettant leur vérification sur un système de règles est introduit et analysé.

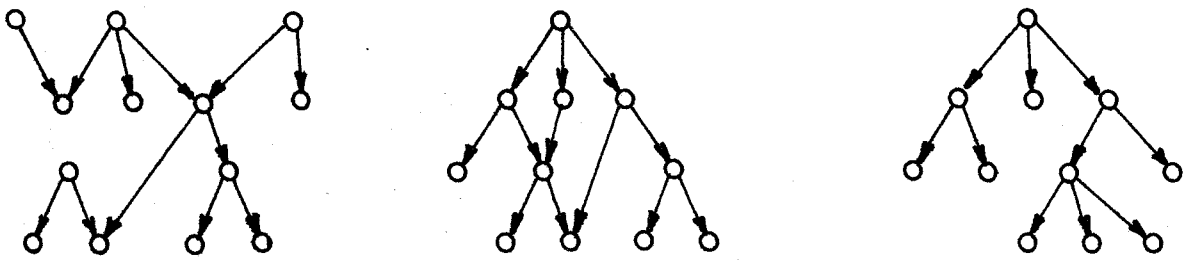
La section 4 de ce chapitre expose divers modèles de coûts d'un processus décisionnel fermé et formule les critères d'optimisation relatifs à ces modèles. On aborde ensuite le problème d'optimisation des PDF, d'abord en présentant un algorithme non déterministe de génération d'arbres de décision et diverses fonctions de choix heuristiques permettant de l'instantier en un algorithme déterministe, puis en discutant deux approches à une optimisation exacte : la programmation dynamique et les procédures de recherche arborescente.

I.2. MODELES DE BASE

I.2.1. ARBRES DE DECISION ET PARTITIONS SUR UN HYPERCUBE

Rappelons quelques définitions :

- un arbre est un graphe connexe sans cycle ;
- un latticiel est un graphe quasi-fortement connexe inférieurement sans circuit, i.e., un graphe connexe muni d'une racine unique et sans circuit ;
- une arborescence est un graphe quasi-fortement connexe inférieurement sans cycle, i.e., un arbre muni d'une racine unique.



arbre latticiel arborescence

FIGURE 1.1.

L'arborescence est une structure particulièrement riche en applications. On la retrouve entre autres dans les problèmes d'énumération, de classification, d'analyse et de recherche de données, en tant que moyen de représentation et de définition (grammaire formelle, monoïde, partitions), en tant que description et modélisation de procédure (inférence logique, optimisation combinatoire), etc... (voir Knuth (121, p.405) pour une présentation générale).

Un abus de langage communément admis restreint la notion d'arbre au seul cas particulier de l'arborescence. Sauf mention explicite, une telle simplification sera faite ici. On adoptera également la terminologie habituelle qui désigne un arc par branche et un sommet par noeud ou feuille suivant qu'il a ou n'a pas de successeur. Seuls les arbres finis seront considérés.

On notera un arbre $t = (V \cup L, \Gamma)$:

- $V = \{ v_1, \ldots, v_p \}$: ensemble des noeuds de t, ordonné par l'ordre scriptural (la racine v_1, suivie de ses fils dans l'ordre gauche - droite, ..., etc) ;

- $L = \{ l_1, \ldots, l_q \}$: ensemble des feuilles de t ;

- Γ : relation d'ordre partiel sur $(V \cup L)$ définissant l'adjacence des sommets de t :

  . $\Gamma(v) = \{ y \in V \cup L \mid (v,y) \text{ est une branche de } t \}$: ensemble des fils de v ;

  . $\hat{\Gamma}$ est la fermeture transitive de Γ, i.e., $\hat{\Gamma}(v)$ = ensemble des descendants de v ;

  . Γ^{-1} est la relation inverse : $\Gamma^{-1}(y)$ est le père de y s'il existe, et $\hat{\Gamma}^{-1}(y)$ est l'ensemble de ses ascendants ($\hat{\Gamma}^{-1}(v_1) = \emptyset$).

Le rang d'un noeud ou d'une feuille de t est :

rang (y) = $| \hat{\Gamma}^{-1}(y) |$, i.e., le nombre de branches du chemin (unique) reliant la racine de t à y.

Un arbre de décision est un outil qui permet de sélectionner un élément particulier dans un ensemble fini donné. Pour ce faire, il associe à la structure arborescente la sémantique suivante :

- à un noeud correspond une condition ou test sur un ensemble de données ;

- les branches issues d'un noeud v définissent une partition du domaine d'application de la condition associée à v ;

- à une feuille est associée une étiquette unique prise dans un ensemble donné. L'arbre est traversé séquentiellement de la racine jusqu'à une feuille. En tout noeud rencontré, la condition correspondante est évaluée et sa valeur détermine la branche unique par laquelle le parcours sera poursuivi.

Soit un système décrit par un vecteur d'état e dans un espace E. Un arbre de décision sur ce système est défini formellement par le 7-tuple

$$t = \{ A, X, V, L, \Gamma, \mathcal{L}, \mathcal{V} \} \qquad \text{où :}$$

- $A = \{ A_1, \ldots, A_a \}$ est un ensemble de a étiquettes distinctes appelées <u>actions</u> ou décisions ; chaque A_j pouvant être une séquence d'actions élémentaires ordonnées $A_j = \{ \alpha_{j_1}, \ldots, \alpha_{ji} \}$;

- $X = \{ X_1, \ldots, X_n \}$ est un ensemble de n fonctions discrètes définies dans E. On dira que $X_i : E \rightarrow Z_i = \{ 0, 1, \ldots z_i - 1 \}$, $z_i \geq 2$, est une <u>fonction caractéristique</u> du système décrit par e, et que l'hypercube correspondant au produit Cartésien $(\overset{n}{\underset{i=1}{\times}} Z_i)$ est <u>l'espace de représentation</u> de ce système ;

- $(V \cup L, \Gamma)$ est un arbre vérifiant $q = |L| \geq a$, et $\forall v \in V : |\Gamma_{(v)}| \geq 2$;

- $\mathcal{L}$ est une application surjective de L dans A, associant à chaque feuille de l'arbre une action unique dans A ; et

- $\mathcal{V}$ est une application de V dans X, associant à chaque noeud v une condition X_i tel que :

  . $\forall v \in V$, $\Gamma_{(v)}$ définit une <u>partition</u> du domaine Z_i de la caractéristique $X_i = \mathcal{V}_{(v)}$, chaque branche issue de v étant associée à un sous-ensemble de Z_i ; et

. $\forall l \in L$, $\mathcal{V}(\hat{\Gamma}^{-1}(l)$ est un ensemble sans répétition, i.e., les caractéristiques associées aux noeuds du chemin (v_1, l) sont toutes distinctes.

A tout état $e \in E$ du système est associé un point unique $(X_1(e), \ldots, X_n(e))$ de l'espace de représentation $(\overset{n}{\underset{i=1}{X}} Z_i)$. A ce point correspondent un chemin et une feuille dans l'arbre t, ainsi qu'une action dans A uniques. Un arbre de décision est donc une structure définissant d'une part une partition de l'espace de représentation d'un système, et d'autre part, une application des classes de cette partition dans un ensemble A. La donnée d'un tel couple (partition sur $\overset{n}{\underset{i=1}{X}} Z_i$; application dans A) ne définit pas néanmoins un arbre unique, mais une classe d'arbres de décision équivalents.

Développons cette équivalence dans le cas particulier où pour tout noeud v de l'arbre tel que $\mathcal{V}(v) = X_i$, v a exactement z_i branches, une branche pour chaque valeur possible de X_i. Introduisons pour cela le formalisme et les propriétés des partitions sur un hypercube.

L'hypercube est une structure qui permet une représentation commode en algèbre de Boole. Les notions de sommets élémentaires, d'arêtes et de faces, ainsi que les relations d'adjacence définies dans le cas booleen sur $\{0, 1\}^n$ sont ici généralisées à l'hypercube $(\overset{n}{\underset{i=1}{X}} Z_i)$.

Soit U l'ensemble de tous les $\prod_{i=1}^{n} (1 + z_i)$ r-cubes (r = 0,1, ..., n) définis sur $(\overset{n}{\underset{i=1}{X}} Z_i)$. Un élément de U est noté vectoriellement $u = (u(1), \ldots, u(i), \ldots, u(n))$; et $u(i) \in \{0, 1, \ldots, z_i - 1\} \cup \{\varphi\}$.

Un 0-cube est un sommet élémentaire qui ne possède aucune composante égale à φ. L'union de z_i sommets élémentaires adjacents le long de leur $i^{ème}$ coordonnée est une arête ou 1-cube :

$$\bigcup_{j=0}^{z_i-1} (u(1),\ldots, u(i-1), j, u(i+1),\ldots, u(n)) = (u(1),\ldots, u(i-1), \varphi, u(i+1),\ldots,u(n))$$

φ étant la composante non spécifiée

On notera :

- $\Phi(u) = \{1 \leq i \leq n \mid u(i) = \varphi\}$; si u est un r-cube, alors $|\Phi(u)| = r$, et u est équivalent à l'union des $(\Pi_{i \in \Phi(u)} z_i)$ sommets élémentaires obtenus en remplaçant dans u chaque coordonnée $u(i) = \varphi$, de toutes les manières possibles par 0, 1, ..., $z_i - 1$;

- $u_o = (\varphi, \varphi, \ldots, \varphi)$: n-cube entier, équivalent à tout $(\bigtimes_{i=1}^{n} Z_i)$;

- $(u/i \leftarrow j) = (u(1), \ldots, u(i-1), j, u(i+1), \ldots u(n))$: r-cube obtenu en fixant la $i^{ème}$ coordonnée de u à j, $j \in Z_i \cup \{\varphi\}$.

On utilisera comme synonyme de r-cube, les mots <u>impliquant</u> ou <u>terme</u>, sans faire de distinction particulière entre l'élément u, et l'ensemble équivalent de sommets élémentaires de l'hypercube : $u \subset (\bigtimes_{i=1}^{n} Z_i)$.

La relation d'inclusion entre r-cubes ($u, v \in U$, $u \subset v$ si $\forall i \in \{1, \ldots, n\}$: $u(i) = v(i)$ ou $v(i) = \varphi$) donne à $(U, \subset)$ une structure d'ordre partiel. Une borne supérieure pour cet ordre est $u_o = (\varphi, \ldots, \varphi)$, une borne inférieure est obtenue par l'adjonction à U de r-cube vide $\emptyset$.

Ainsi augmenté, U est un demi-treillis inférieur, avec $\text{Inf}(u,v) = u \cap v = w$, tel que :

- si $\exists$ i tel que $u(i) \neq v(i)$, et $u(i) \neq \varphi$, et $v(i) \neq \varphi$; alors $w = \emptyset$ (u et v sont dits indépendants ou sans recouvrement),

- sinon $\forall i \in \Phi(u) : w(i) = u(i)$, et $\forall i \notin \Phi(u) : w(i) = v(i)$.

On remarque que :

- le demi-treillis inférieur correspond à un latticiel Dedekindien de racine u_o <u>i.e.</u>, un latticiel où tous les chemins entre 2 sommets quelconques ont le même nombre d'arcs (Cf. figure 1.2. pour un exemple correspondant à $\{0,1,2\} \times \{0,1\}^2$);

- sauf dans le cas booleen où U est un treillis complet, l'opération Sup (u,v) = u ∈ v ne définit pas en général un élément, mais un sous-ensemble de U.

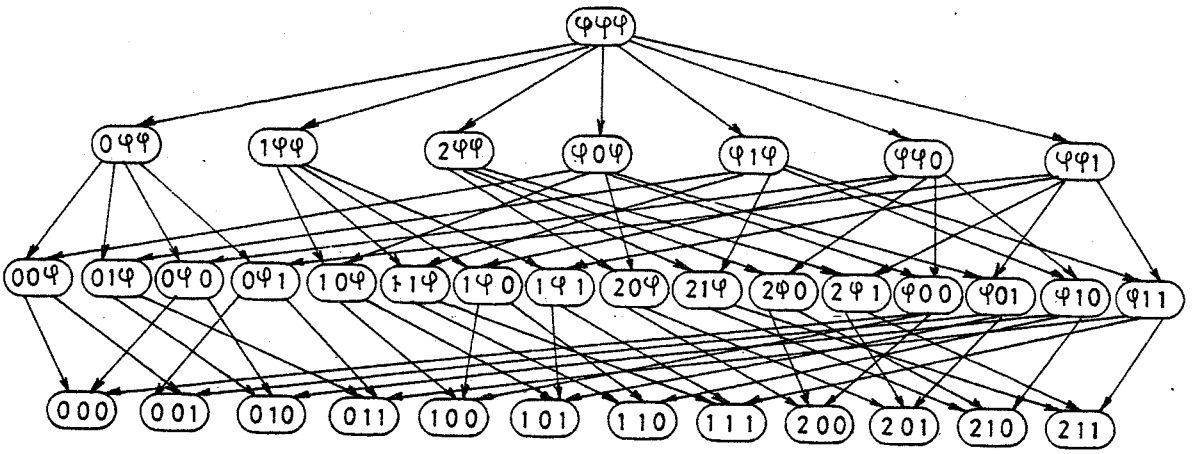


FIGURE 1.2.

Considérons l'ensemble π des partitions sur les sommets de $(\overset{n}{\underset{i=1}{X}} Z_i)$ et la relation d'ordre partiel : π_1 est une partition plus fine que π_2 (notée $\pi_1 \leq \pi_2$) si tout bloc de π_1 est inclus dans un bloc de π_2. $(\pi, \leq)$ possède également une structure de treillis dont le plus petit élément est la partition de $(\overset{n}{\underset{i=1}{X}} Z_i)$ en sommets élémentaires.

La borne inférieure de deux partitions est leur partition produit. La borne supérieure de π_1 et π_2 est la partition π telle que : 2 sommets sont équivalents modulo π s'il existe une suite d'équivalences, chacune étant soit modulo π_1, soit modulo π_2, qui permette de passer d'un sommet à l'autre.

Une partition π est dite homogène si chacun de ses blocs est un r-cube de U. On a la :

Proposition 1.1. : la partition de l'hypercube définie par un arbre de décision t est une partition homogène.

Preuve : résulte simplement des faits :

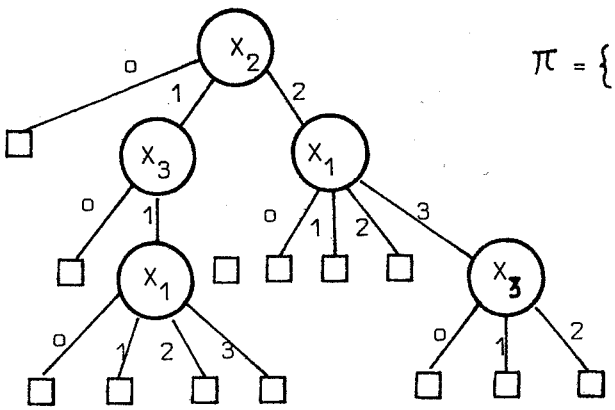
1) que la partition définie par t possède un bloc par feuille l de t ;

2) que les caractéristiques associées à $\hat{\Gamma}^{-1}(l)$ sont toutes distinctes, et

3) qu'une branche issue d'un noeud v correspond à une valeur unique de la caractéristique $X_i = \mathcal{V}(v)$. □

On note $(l_1, \ldots, l_q)$ la partition définie par l'arbre t, la feuille l de rang k étant identifiée à un (n-k)-cube tel que : $l(i) = j$ si $X_i \in \mathcal{V}(\hat{\Gamma}^{-1}(l))$ et le chemin de la racine de t à l emprunte la $j^{ème}$ branche de X_i, sinon $l(i) = \varphi$.

Exemple 1.1. : La partition π définie par l'arbre ci-dessous sur $\{0, 1, 2, 3\} \times \{0, 1, 2\}^2$ est :



$\pi = \{ (\varphi o \varphi), (\varphi 10), (\varphi 12), (011), (111), (211), (311)$
$(02\varphi), (12\varphi), (22\varphi), (320), (321), (322) \}$

FIGURE 1.3.

A toute partition homogène correspond-il un arbre de décision qui la définisse ? La réponse est négative, comme le montre le contre-exemple suivant (Fig. 1.4) d'une partition sur l'hypercube $\{0,1,2\}^3$.

Exemple 1.2. : $\pi = \{ (\varphi 00), (\varphi 02), (\varphi 20), (\varphi 22), (0\varphi 1), (2\varphi 3), (11\varphi), (010), (012), (101), (102), (210), (212) \}$

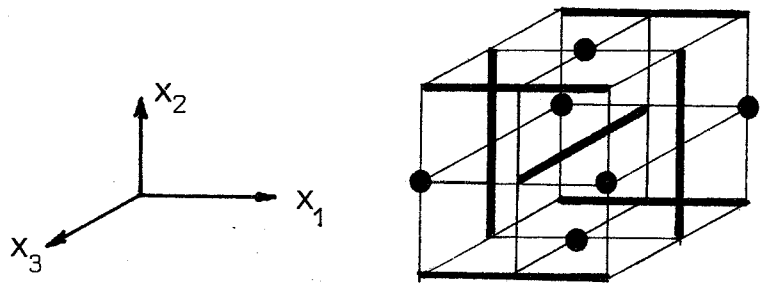


FIGURE 1.4.

L'existence d'au moins une coordonnée explicite (i.e., différente de "φ") dans tous les termes d'une partition homogène π est une condition nécessaire (mais non suffisante) pour que π soit définissable par un arbre. On note Π_T le sous-ensemble des partitions définissables par un arbre, et Π_H celui des partitions homogènes. On a : $\Pi_T \subset \Pi_H \subset \Pi$.

Deux partitions de Π_H (ou de Π_T) n'ont pas nécessairement une borne supérieure dans Π_H (resp. dans Π_T). Ces deux sous-ensembles ne sont donc pas des sous-treillis de $(\Pi, \leq)$, mais on peut s'y ramener moyennant certaines extensions (Cf. par exemple Yasui (244)).

Soit π une partition quelconque de $(\bigtimes_{i=1}^{n} Z_i)$ en a blocs, chaque bloc étant étiqueté d'une façon biunivoque par une des actions de $A = \{A_1, \ldots, A_a\}$; et soit t un arbre de décision définissant sur $(\bigtimes_{i=1}^{n} Z_i)$, la partition $(l_1, \ldots, l_q)$.
On dira que :

- l'arbre t <u>réalise</u> la partition π si $(l_1, \ldots, l_q) \leq \pi$, et pour toute feuille l incluse dans un bloc de π associé à A_j : $\mathcal{L}(l) = A_j$;

- deux arbres t et t' sont <u>faiblement équivalents</u> modulo π s'ils réalisent tous deux cette partition π ;

- t et t' sont <u>fortement équivalents</u> s'ils définissent une même partition sur $(\bigtimes_{i=1}^{n} Z_i)$ et une même application dans A.

Exemple 1.3. : deux arbres fortement équivalents définissant la partition $\pi = \{(01\varphi 0), (110\varphi), (10\varphi\varphi), (00\varphi\varphi), (01\varphi 1), (111\varphi)\}$ sur $\{0, 1\}^4$:

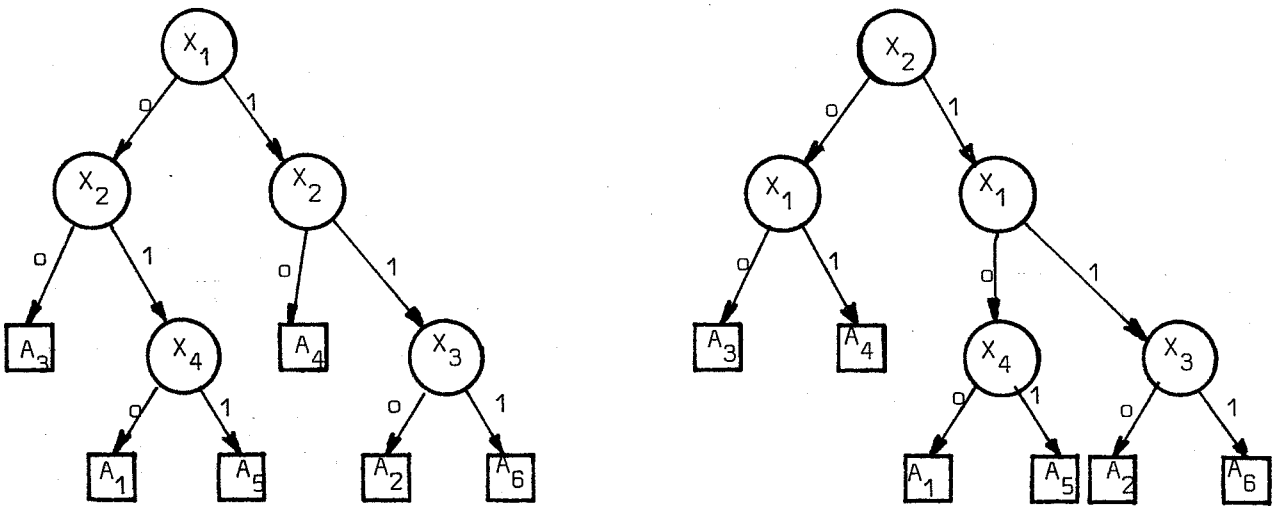


FIGURE 1.5.

I.2.2. PROCEDURES D'IDENTIFICATIONS ET QUESTIONNAIRES

De nombreuses applications dans les domaines du diagnostic, du test et de la détection de pannes, en informatique pour des problèmes de codage ou de recherche et de tri dans un ensemble, ou en reconnaissance de forme dans certains cas d'identification et de classification automatique, ont stimulé le développement d'outils appelés suivant les auteurs : Questionnaires (37,184) ou Procédures d'identifications (57,58) .

Ces diverses applications font en effet intervenir un même problème central que l'on peut formuler ainsi : comment isoler un élément inconnu dans un ensemble par une succession de partitions de cet ensemble et d'identifications des classes où figure l'élément recherché ?

Le modèle idéal, que la théorie des questionnaires s'est initialement attachée à développer, suppose que toutes les partitions d'une certaine cardinalité sont utilisables. Plus réaliste pour la plupart des applications citées, le modèle des Procédures d'identification et des Questionnaires réalisables(33-138) s'efforce d'isoler l'élément inconnu au moyen de partitions admissibles données à priori.

Dans ce modèle, on formalise le problème d'identification par :

. $\Theta = \{\theta_1, \ldots, \theta_m\}$ un ensemble fini d'événements ou d'états mutuellement exclusifs, parmi lesquels figure l'état inconnu θ à identifier ;

. $Q = \{Q_1, \ldots, Q_n\}$ un ensemble d'applications surjectives de Θ dans des domaines finis d'entiers. Chaque application $Q_i : \Theta \rightarrow \{0, 1, \ldots, z_i - 1\}$ définit une partition de Θ en z_i classes ; elle est dite test ou question sur Θ. Poser la question Q_i pour identifier l'état inconnu θ revient à déterminer la classe $Q_i(\theta)$ où figure cet état.

La matrice (n x m) d'élément générique $Q_i(\theta_j)$, appelée table d'identification, résume l'ensemble des données d'un problème d'identification.

Exemple 1.4. : Un système (idéalisé) de tri d'objets dans une chaîne robotisée doit reconnaître une pièce mécanique (isolée) appartenant à l'ensemble suivant :

Θ = { rondelle, écrou, vis, plaque, joint, raccord } (FIGURE 1.6.)

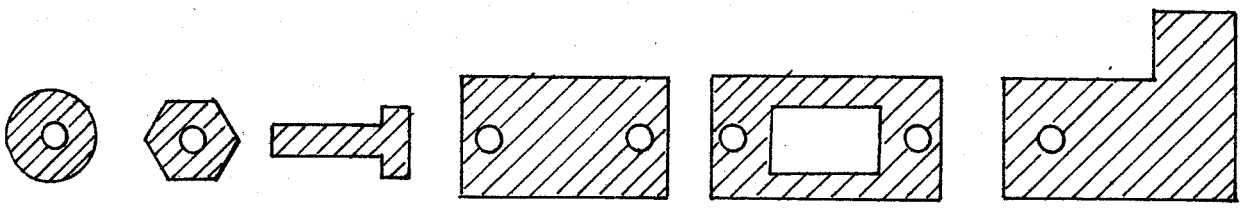


FIGURE 1.6. : Objets à trier

Le système sait répondre aux trois questions suivantes :

Q_1 : quel est le nombre de trous de l'objet perçu ?

Q_2 : le contenu externe de l'objet est-il circulaire, rectangulaire, ou a-t-il une autre forme ?

Q_3 : le périmètre externe de cet objet est-il petit, moyen ou grand ?

Les partitions définies pour ces 3 questions sur Θ sont données par la table d'identification (fig. 1.7.) :

	rondelle	écrou	vis	plaque	joint	raccord
Q_1	1	1	0	2	3	2
Q_2	circulaire	autre	autre	rectangulaire	rectangulaire	autre
Q_3	petit	petit	moyen	grand	grand	grand

FIGURE 1.7. : Table d'identification

Deux événements θ_j et θ_k seront dits inséparables si les colonnes j et k de la table d'identification sont identiques. De tels événements doivent être regroupés en un événement unique, ce que nous supposerons par la suite. Si deux questions définissent la même partition sur Θ (à une permutation des indices des classes près), alors l'une est redondante et peut être supprimée.

Chaque événement θ_j du problème d'identification peut être considéré comme un sommet élémentaire de l'hypercube $(\overset{n}{\underset{i=1}{X}} \{0, 1, \ldots, z_i - 1\})$, dont les coordonnées sont la $j^{ème}$ colonne $(Q_1(\theta_j), \ldots, Q_n(\theta_j))$ de la table d'identification. Cette table est donc une partition en (m+1) blocs : un bloc par sommet élémentaire associé à un événement de Θ, et un $(m+1)^{ème}$ bloc pour tous les autres sommets considérés implicitement comme des états "impossibles".

En effet dans un problème d'identification, on suppose que seuls sont possibles les événements explicités de Θ. Les autres combinaisons des valeurs des questions traduisent des relations de dépendances entre celles-ci. Ainsi dans l'exemple précédent, on exclut à priori la possibilité de rencontrer un objet circulaire à plus d'un trou, ou à grand périmètre. Le fait de savoir qu'un objet est circulaire suffit pour identifier une rondelle. D'où les définitions :

- Une partition homogène π est admissible pour le problème d'identification si tout bloc de π contient au plus un événement unique de Θ ;

- Une procédure d'identification ou questionnaire sur un problème d'identification est un arbre de décision définissant une partition admissible pour ce problème. Dans un tel arbre $A = \Theta$ et $\mathcal{L}(1) = \theta_j$ si l'événement θ_j appartient au bloc défini par la feuille 1. Une feuille ne contenant aucun événement est élaguée de l'arbre.

La figure(1.8.) est un questionnaire sur le problème de l'exemple 1.4.

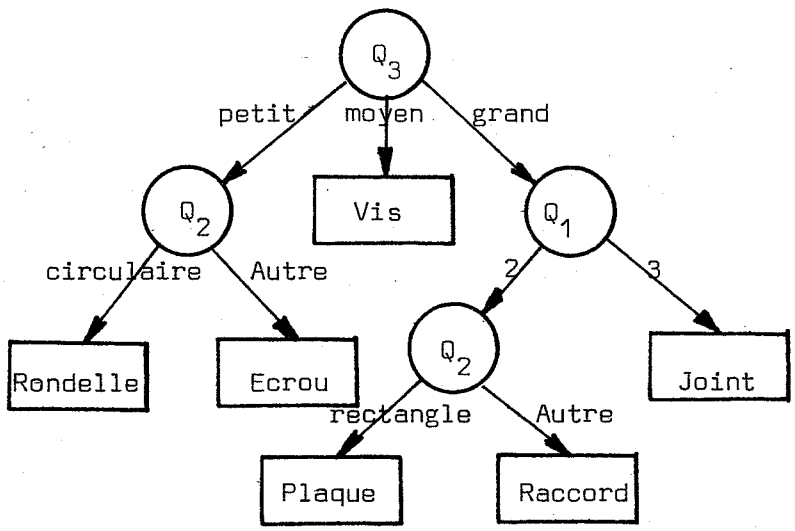


FIGURE 1.8. : Une procédure d'identification

Remarquons que du fait de l'élagage des feuilles "impossibles" :

- un questionnaire a autant de feuilles que d'événements, et $\mathcal{L}$ est une bijection entre L et Ⓗ ;

- un noeud correspondant à une question Q_i n'a pas nécessairement z_i branches, et le questionnaire ne définit plus qu'une partition d'un sous-domaine de l'hypercube incluant tous les événements de Ⓗ .

Le modèle des questionnaires, tel qu'il a été brièvement exposé, semble extrêmement restreint. Deux types de généralisations portant sur la nature des questions ont été abordés dans la littérature :

- en supposant toutes les questions d'une certaine cardinalité disponibles, Picard (185,186) a étudié le cas où chaque question est un recouvrement et non une partition de Ⓗ . Le questionnaire devient alors un latticiel.

- dans les pseudo-questionnaire développés par Terrenoire et Chenais (23,233) à une question Q_j sont associées une distribution de probabilités discrètes, chacune fournissant la probabilité conditionnelle de Q_j sachant un événement θ_i. L'identification de l'état inconnu repose alors sur une classification Bayesienne .

I.2.3. TABLES DE DECISION

Etudiées initialement en parallèle avec des outils de synthèse de fonctions booléennes (19) , puis comme moyen assez général d'analyse et de traitement d'information logique, les tables de décision ont connu un développement important et continuent d'être largement utilisées. Un grand nombre d'ouvrages et de publications leur ont été consacrés, parmi lesquels (96,142,153,159,190,192) sont quelques livres de base et articles de synthèse.

Le principal domaine d'application des tables de décision réside dans l'analyse, la documentation, la vérification et la conception de logiciels. Dans ce domaine, les tables offrent des avantages de simplicité, de clarté, de facilité d'élaboration et de modification, et surtout de nombreuses possibilités de test et de manipulations automatiques.

Certains systèmes, pour la certification et la preuve de programme, transforment ceux-ci en tables de décision (9,20) . D'autres systèmes plus nombreux font la transformation inverse, en optimisant éventuellement le programme résultant et en testant au préalable la structure logique de la table.

Sur un plan théorique, Larson(132)a montré qu'une table de décision est d'une part équivalente à une machine de Turing, et, d'autre part, qu'elle peut représenter tout processus modélisable par un réseau de Petri. Sur un plan pratique, le programmeur désireux de s'exprimer sous forme de tables de décision trouvera sur le marché plus de 50 "packages" différents allant de préprocesseurs pour de nombreux langages (y compris LISP(219)ou APL(132)), jusqu'aux compilateurs intégrés à des systèmes de programmation complets (152,159).

A côté des applications à la programmation et aux problèmes qui en sont proches, tels que ceux de recherche et d'organisation de fichiers, ou des problèmes de simulation, les tables de décision ont également été utilisées dans toutes les applications précédemment citées pour les questionnaires : inspection, diagnostic, tri, reconnaissance de forme (12,220) ; et récemment, on les trouve dans certains "systèmes experts" en intelligence artificielle (234).

De nombreuses définitions formelles des tables de décision ont été proposées. Cavouras (20)en cite 5 équivalentes, allant des équations conditionnelles de Mc Carthy(190)aux fonctions partielles sur un domaine fini (197), définition étendue aux fonctions récursives dans (164,132).

Informellement, une table de décision consiste en la description d'un ensemble de conditions (traditionnellement des propositions logiques), d'un ensemble d'actions élémentaires et d'une correspondance entre les valeurs des conditions et les actions à entreprendre.

Exemple 1.5. : La structure tabulaire est habituellement celle de la figure 1.9(a). La partie gauche décrit les 4 conditions binaires de la table X_1, X_2, X_3, X_4, et les trois actions α_1, α_2, α_3 ; la partie droite précise la correspondance entre les valeurs des conditions et des actions. Ainsi, la 1ère colonne s'interprète :

Si (X_1 = 1 et X_2 = 1 et X_3 = 0 et quelle que soit la valeur de X_4)
Alors effectuer les actions (α_1 et α_3).

X_1	1	0	φ	0	φ	0
X_2	1	1	1	φ	0	0
X_3	0	φ	1	0	0	1
X_4	φ	1	0	0	1	φ
α_1	*	*	*		*	
α_2		*		*	*	*
α_3	*		*			

(a)

X_1	1	φ	0	φ	0	0
X_2	1	1	1	0	φ	0
X_3	0	1	φ	0	0	1
X_4	φ	0	1	1	0	φ
	A_1		A_2		A_3	

(b)

FIGURE 1.9.

On remarque dans la figure 1.9(a) des colonnes identiques dans la partie actions, qui sont regroupées dans la représentation équivalente (b) avec : $A_1 = (\alpha_1, \alpha_3)$; $A_2 = (\alpha_1, \alpha_2)$; et $A_3 = (\alpha_2)$.

Dans certaines tables de décision, l'ordre d'exécution des actions élémentaires dans une séquence est explicité pour chaque colonne. Dans une autre variante, la table comporte une règle Autre, i.e. une séquence d'actions à effectuer si les valeurs des conditions ne correspondent à aucune autre séquence. Pour les applications à la programmation, deux types d'actions élémentaires sont distinguées les actions agissant sur les données et les actions de contrôle qui permettent de relier entre elles plusieurs tables et d'introduire éventuellement des récursions.

Plusieurs variantes interviennent également dans l'interprétation d'une table de décision. Certains auteurs(119) admettent que la table puisse spécifier plusieurs séquences d'actions pour une même combinaison de valeurs des conditions, auquel cas il faudrait les exécuter toutes. Cette interprétation rejoint celle des "grilles de décision" (Decision Grid Charts), lesquelles peuvent être transformées par regroupement en tables de décision, où à toute combinaison de valeurs des conditions correspond une séquence d'actions unique(144,230). Une table qui ne vérifie pas cette propriété est alors incomplète (pas d'action prévue pour certains cas), ou ambiguë (recouvrement de plusieurs séquences d'actions).

Traditionnellement, les conditions d'une table de décision sont supposées être des propositions logiquement indépendantes. Un algorithme simple peut alors être utilisé pour détecter les ambiguïtés dans une table(190). La notion de dépendance entre conditions discutée par King (116-118) permet de distinguer, lors de la vérification d'une table, les ambiguïtés réelles des ambiguïtés apparentes.

Plusieurs méthodes ont été proposées pour mettre en oeuvre une table de décision sur calculateur. La plus simple consiste à tester successivement toutes les conditions et à incrémenter, suivant le résultat de chaque test, un compteur qui fournit en fin de procédure l'adresse de la séquence d'actions à exécuter (238). Cette méthode du type " Hash table" est très coûteuse relativement au nombre de conditions testées. La méthode des masques explore la table de décision colonne par colonne, et ne teste pour chacune que les conditions non encore testées nécessaires pour éliminer ou retenir éventuellement la séquence d'actions correspondantes.

Malgré quelques heuristiques proposées pour ordonner les colonnes de la table en vue de réduire les tests, cette approche reste peu performante (27,116,165) . Si l'on désire optimiser l'implémentation d'une table, il faut avoir recours à la méthode des arbres de décision. On reviendra ultérieurement à cette méthode et à la littérature importante qui lui a été consacrée.

La principale extension apportée par le modèle des tables de décision relativement à celui des tables d'identification provient du fait que ce dernier ne considère que des sommets élémentaires de l'espace de représentation, alors que les colonnes d'une table de décision portent sur des sous-ensembles de r-cubes:

certaines conditions y sont "indifférentes" (coordonnées Ψ). Il s'ensuit qu'une question est une partition sur les événements d'un questionnaire, alors qu'une condition n'est qu'un recouvrement des séquences d'actions d'une table. Par ailleurs, un problème d'identification est implicitement complet et non ambigu : tout événement qui ne figure pas dans Θ est impossible. En conséquence, la notion de règle Autre , qui présente des avantages pour de nombreuses applications, est indéfinissable sur un questionnaire.

Le modèle de processus décisionnel fermé pour un système de règles de décision, exposé dans la section suivante, est assez proche dans sa formulation initiale de celui des tables de décision. Néanmoins, il permettra de prendre en compte certaines généralisations sur les caractéristiques (relations de dépendance, contraintes de succession et valeurs indifférentes). Par ailleurs, le formalisme des systèmes de règles de décision permettra une présentation unifiée des processus décisionnels ouverts et fermés.

°0°

I.3. FORMULATION D'UN MODELE GENERAL ET PRINCIPALES PROPRIETES

On définira un système de règles de décision sous forme généralisée, puis sous forme normale. Les propriétés de consistance et de complétude seront introduites. Un processus décisionnel fermé pour système de règles sera défini en terme de procédure récursive interprétant un arbre de décision associé aux partitions d'un hypercube.

I.3.1. SYSTEME DE REGLES DE DECISION

Une condition sur un système est un prédicat du type $(X_i(e) \in k_i)$ avec :

- $e \in E$; E espace d'état du système ;
- $X_i : E \rightarrow K_i$; fonction caractéristique du système à valeurs dans un domaine K_i, ensemble quelconque, et $k_i \subset K_i$.

Périodiquement, ou à la suite d'un événement particulier, un automate est amené à exécuter une séquence d'actions prises dans un ensemble $\{\alpha_1, \ldots, \alpha_r\}$ fini et donné d'actions élémentaires, et qui dépend de l'état e du système à ce moment.

Ce système est dit gouverné par un ensemble de règles de décision, ou système de règles de décision, si l'on dispose d'un ensemble fini de règles du type :

"Si ($\mathcal{F}_j$) , alors (A_j)" où :

- $\mathcal{F}_j$ est une expression bien formée (ebf) de logique propositionnelle portant sur un ensemble de conditions liées par les connecteurs usuels $\wedge$, v, $\neg$;
- A_j est une séquence ordonnée d'actions élémentaires prises dans l'ensemble $\{\alpha_1, \ldots, \alpha_r\}$.

On dit que ($\mathcal{F}_j$) est l'antécédent de la règle et (A_j) est son conséquent.

Une telle règle précise que si au moment de la décision, le système se trouve dans un état e pour lequel $\mathcal{F}_j$ est vrai, alors A_j est la séquence d'actions à exécuter. Dans un premier temps, on ne s'intéressera pas à la modification de l'état du système résultant de l'exécution de A_j. On admet également la possibilité de spécifier une règle "Autre" qui précise la séquence d'actions à exécuter si aucune autre règle n'a son antécédent ($\mathcal{F}_j$) vrai.

En général, les diverses fonctions caractéristiques du système ne sont pas indépendantes. Plusieurs types de relations de dépendances sont possibles, mais on ne considèrera initialement que le type logique, i.e., les dépendances exprimables sous les formes :

$$\forall e\ \mathcal{F}\ ; \quad \text{ou} \quad \forall e\ (\mathcal{F} \Rightarrow \mathcal{G})\ ; \quad \text{ou} \quad \forall e\ (\mathcal{F} \Leftrightarrow \mathcal{G})$$

$\mathcal{F}$ et $\mathcal{G}$ étant des ebf sur un ensemble de conditions.

On cherchera à ramener un système de règles sous forme normale.

Cette représentation passe par :

1) la segmentation du domaine des fonctions caractéristiques ;

2) le regroupement des règles menant à des séquences d'actions identiques ;

3) la transformation des antécédents des règles sous forme normale disjonctive ; et

4) la transformation des relations de dépendances de tautologies en contradictions sous forme normale disjonctive.

Le domaine d'application des caractéristiques est discrétisé et segmenté par un produit de partitions. Si $k_{i_1}, \ldots, k_{i_s}$ sont tous les sous-domaines de K_i qui interviennent dans les diverses conditions où figure la caractéristique X_i, on construit la partition produit des bi-partitions (k_{ij} ; $K_i - k_{ij}$). On regroupe ensuite en une seule classe tous les blocs de ce produit qui ne figurent dans aucun k_{ij}.

Les z_i blocs de la partition résultante sont indicés sur $Z_i = \{0,1, \ldots, z_i-1\}$ qui devient le nouveau domaine d'application de X_i. Toute condition $(X_i(e) \in k_i)$ est alors remplacée par la disjonction : $(X_i(e) = j_1) \vee \ldots \vee (X_i(e) = j_r)$; $j_1, \ldots, j_r$ étant les indices des blocs inclus dans k_i.

Soit $A = \{A_1, \ldots, A_a\}$ l'ensemble des diverses séquences d'actions élémentaires spécifiées par le système de règles. Les règles menant à la même séquence A_i sont regroupées en une règle unique :

"Si $(\mathcal{F}_{i_1}) \vee \ldots \vee (\mathcal{F}_{i_j})$, alors (A_i)"

L'antécédent de chacune de ces a règles est ramené sous forme normale disjonctive par les transformations classiques suivantes :

- suppression des opérateurs de négation (report de chaque opérateur au niveau d'une condition, et remplacement de $\neg (X_i(e) = j)$ par la disjonction complémentaire) ;

- mise sous forme normale disjonctive dont chaque terme est considéré comme un r-cube sur $(\overset{n}{\underset{i=1}{X}} Z_i)$: au terme $(X_{i_1}(e) = j_1) \wedge \ldots \wedge (X_{i_k}(e) = j_k)$ correspond u tel que : $u(i_1) = j_1$; ... ; $u(i_k) = j_k$; et $u(i) = \varphi$ pour $i \notin \{i_1, \ldots, i_k\}$.

Remarquons que la représentation obtenue n'est pas unique : on peut procéder à des regroupements de monômes et, éventuellement, à une minimisation.

Les mêmes transformations s'appliquent aux relations de dépendances ramenées au préalable à des contradictions.

Finalement, un système de règles sous forme normale sera représenté par le 4-tuple $S = \{X, A, D, R\}$:

- $X = \{X_1, \ldots, X_n\}$, $X_i : E \rightarrow Z_i = \{0, 1, \ldots, z_i-1\}$;

- $A = \{A_1, \ldots, A_a\}$;

- $R = \{R_1, \ldots, R_a\}$ ensemble des règles de décision de S ; la règle R_j associée à

l'action A_j est considérée comme équivalente à l'ensemble des relations qui interviennent dans son antécédent ;

- D : relation de dépendance entre les caractéristiques du système, également équivalente à un ensemble de r-cube $\{u_1, \ldots, u_d\}$. Chacun étant une conjonction de conditions fausses pour tout état $e \in E$.

Remarque : Chaque règle R_i ou D est à la fois un sous-ensemble de termes de U et un sous-ensemble de sommets élémentaires de $(\bigtimes_{i=1}^{n} Z_i)$, les deux structures n'étant distinguées que par le contexte.

I.3.2. CONSISTANCE ET COMPLETUDE D'UN SYSTEME DE REGLES DE DECISION

Un système de règles S est complet si pour tout état $e \in E$, il existe au moins une règle R_j dont l'antécédent est vrai pour e. Le système S est consistant si cette règle est unique.

La vérification de la consistance et de la complétude fait intervenir deux types de problèmes, l'un sémantique et l'autre syntaxique. Au niveau sémantique, on suppose que les relations de dépendance ne sont pas connues ou ne le sont que partiellement. Il s'agit alors, à partir d'une description de l'ensemble des caractéristiques, de déterminer D, ou de le compléter et de le vérifier.

Dans le cas particulier où chaque caractéristique est une forme linéaire sur le vecteur d'état e, on peut utiliser des méthodes algébriques. En se restreignant à des caractéristiques binaires ne faisant intervenir que des inégalités de formes linéaires, Ibramsha et Rajaraman(103)ont développé un algorithme polynômial qui permet de vérifier que l'ensemble des contraintes correspondantes admet au moins une solution.

Dans le cas où les caractéristiques sont décrites en termes de prédicats du premier ordre, on pourra vérifier (principe de résolution) qu'un terme de D correspond bien à une contradiction si tel est le cas, mais on ne pourra pas générer ou compléter D (semi-décidabilité des théories du premier ordre). Dans le cas général, le problème de la vérification de la consistance et de la complétude est indécidable.

Au niveau syntaxique, on suppose que D exprime toutes les dépendances du système ; il ne reste alors à vérifier que l'ensemble des règles. On a les conditions nécessaires et suffisantes suivantes :

Proposition 1.2. : S est un système complet ssi $R \cup D = (\overset{n}{\underset{i=1}{\times}} Z_i)$;

S est consistant ssi $\forall i,\ j \neq i : R_i \cap R_j \subset D$.

Preuve : A tout état $e \in E$, il correspond un terme $u = (X_1(e), \ldots, X_n(e))$ qui, par hypothèse, n'appartient pas à D. Si $R \cup D = (\overset{n}{\underset{i=1}{\times}} Z_i)$, alors il existe une règle R_i tel que $u \in R_i$, et donc S est complet. Réciproquement, la complétude entraîne que pour tout $u \in (\overset{n}{\underset{i=1}{\times}} Z_i)$, $u \notin D$, il existe une règle R_i tel que $u \in R_i$ et donc $R \cup D = (\overset{n}{\underset{i=1}{\times}} Z_i)$. □

La condition sur la consistance s'établit aussi simplement.

La vérification de la complétude d'un système de règles est un problème équivalent à la réalisation d'une expression de logique propositionnelle. En effet, $R \cup D = (\overset{n}{\underset{i=1}{\times}} Z_i)$ si la disjonction de termes $\Omega = \bigvee_{u \in S} u$ est une tautologie, ou d'une manière équivalente, si la négation de Ω n'est pas réalisable. Le problème de la réalisation d'une expression logique est bien connu comme élément générique initial de la classe des problèmes NP-Complets (26) . Un algorithme vérifiant la complétude de S n'évitera donc une complexité exponentielle (si P ≠ NP) que dans le cas particulier où tous les termes de S sont 2 à 2 disjoints. En effet, un comptage des coordonnées (φ) des termes suffit dans ce cas. S est complet si :

$$\sum_{u \in S} \left(\prod_{i \in \Phi(u)} z_i \right) = \prod_{1 \leq i \leq n} z_i$$

Ayant une latitude dans l'expression des règles, en particulier lors de la mise sous forme normale de S, on admettra une hypothèse moins contraignante d'indépendance 2 à 2 des termes d'une même règle, et d'indépendance 2 à 2 des termes de D. Un algorithme de vérification de la consistance et de la complétude de S peut alors être le suivant (Q est un ensemble de termes de U ; q, y et d sont des entiers) :

Algorithme VERIFICATION

Données d'entrée : $S = \{X, A, D, R\}$

1. Initialisation : $Q \leftarrow \emptyset$; $y \leftarrow 0$

2. Itérer sur $\{(u, v) \in R_j \times R_l \mid 1 \leqslant j \leqslant a \ ; \ 1 \leqslant l \leqslant a \ ; \ j \neq l\}$

2.1. Si $(w \leftarrow u \cap v) \neq \emptyset$, alors faire :

2.1.1. $d \leftarrow 0$

2.1.2. Itérer sur $\{w' \in D\}$

Si $w \cap w' \neq \emptyset$, alors $d \leftarrow d + \prod_{i \in \overline{\Phi}(w \cap w')} z_i$

Fin itération 2.1.2.

2.1.3. Si $d = \prod_{i \in \Phi(w)} z_i$, alors $y \leftarrow y + d$ (w est inclus dans D)

2.1.4. Sinon $Q \leftarrow Q \cup \{w\}$

2.1.5. Fin itération 2.

3. $q \leftarrow \prod_{1 \leqslant i \leqslant n} z_i - \left[\sum_{u \in S} \left(\prod_{i \in \Phi(u)} z_i \right) - 2y \right]$

Données de sortie : Q et q.

Proposition 1.3. : S est consistant ssi $Q = \emptyset$; et dans ce cas, S est complet ssi $q = 0$.

Preuve : Par hypothèse, les termes de D sont 2 à 2 disjoints. Donc, à la fin de l'itération (2.1.2.), d est le nombre de sommets élémentaires distincts de w qui figurent dans D ; et $w \subset D$ si $d = \prod_{i \in \Phi(u)} z_i$. Ceci établit la preuve de la consistance. Relativement à la complétude, les sommets de tous les termes $w = u \cap v$ sont comptés trois fois dans $\left[\sum_{u \in S} \left(\prod_{i \in \Phi(u)} z_i \right) \right]$: 1) dans $u \in R_j$; 2) dans $v \in R_1$; et 3) dans D. Tous les autres sommets ne sont comptés qu'une seule fois du fait de l'indépendance des termes d'une règle.

Analyse de l'algorithme : Si m est le nombre total de termes de S, le nombre de couples distincts de l'itération 2 est au plus $m(m-1)/2$. L'itération (2.1.2.) est effectuée d fois, d étant le nombre de termes de D ; et l'intersection de 2 r-cubes nécessite n opérations. La complexité de VERIFICATION est donc en $O(m^2 * n * d)$.

Remarques :

1) Le test (2.1.3.) peut être déplacé et modifié pour, d'une part, réduire l'itération (2.1.2.) à un sous-ensemble de D, et d'autre part, donner la complétude de S indépendamment de sa consistance.

2) Dans le sens d'une mise en oeuvre interactive pour l'aide à la vérification d'un système, la donnée des inconsistances (termes de Q) est essentielle. Celle des incomplétudes, explicitées par un ensemble de termes non couverts par S est également souhaitable, mais elle ne peut être fournie qu'au prix d'une complexité exponentielle. Pour l'éviter, on peut utiliser la règle Autre qui, par définition, rend S complet.

Pour clôre cette section, mentionnons deux autres interprétations possibles pour la consistance d'un système de règles. Si plusieurs règles sont simultanément valides pour un même état e , on peut :

- soit les exécuter toutes ;

- soit en choisir une particulière qui sera exécutée ("Conflict set resolution").

Dans le premier cas, on pourra définir une nouvelle règle correspondant à l'intersection de celles qui sont simultanément valides et se ramener à la définition donnée. Pour certaines applications, la résolution du conflit (choix d'une règle parmi les valides) s'effectue en testant certaines caractéristiques qui pourront éventuellement être ajoutées aux antécédents des règles et permettront de se ramener à un système consistant.

1.3.4. PROCESSUS DECISIONNEL SUR UN SYSTEME DE REGLES

D'une manière générale, par processus décisionnel fermé (PDF) sur un système de règles de décision S, on entend toute procédure qui appelée sur un état $e \in E$ quelconque, fournit la règle de décision unique qui s'applique à cet état. Une telle procédure peut, soit examiner directement le système S, soit porter sur une structure équivalente à ce système obtenue par un prétraitement préalable. En utilisant la terminologie des langages de programmation, on parlera de système interprété ou compilé dans l'un ou l'autre cas.

Une procédure interprétant un système S commencera par évaluer toutes les caractéristiques de S sur l'état e, puis passera en revue les termes des règles de S en s'arrêtant au premier terme u tel que $(X_1(e), \ldots, X_n(e)) \subseteq u$; la règle contenant u est celle qui est valide dans e. On peut améliorer la procédure en examinant les règles terme par terme, et en n'évaluant pour chaque terme u que les caractéristiques qui permettent, soit de confirmer l'inclusion précédente, soit de rejeter u. Cela correspond à l'algorithme suivant :

Algorithme INTERPRETATION

Données d'entrée : $S = \{X, A, D, R\}$; $e \in E$

1. Initialisation : $v \leftarrow u_0$; Action $\leftarrow$ 0.

2. Itérer sur $\{ u \in R_i \mid R_i \in R, \text{ et } u \cap v \neq \emptyset \}$ tant que (Action = 0)

2.1. Itérer sur $\{ j \in [\overline{\Phi}(v) - \overline{\Phi}(u \cap v)] \}$ tant que $(u \cap v \neq 0)$

2.1.1. Evaluer $X_j(e)$

2.1.2. $V \leftarrow (V/j \leftarrow X_j(e))$

2.1.3. Fin itération 2.1.

2.2. Si $V \subseteq u$, alors Action $\leftarrow A_i$

2.3. Fin itération 2

3. Fin

Données de sortie : Action

Il est simple de voir que, pour S consistant, l'algorithme s'arrête en fournissant l'action correspondante à l'unique règle valide du système. Notons que si l'itération (2) porte sur un terme u tel que $v \subset u$, alors l'itération (2.1.) n'a pas lieu car, dans ce cas, $\overline{\Phi}(u) = \overline{\Phi}(u \cap v)$. Ne sont donc testées en plus sur chaque terme u que les caractéristiques indifférentes de V qui sont explicitées dans u. Néanmoins, comme on le verra plus loin, l'évaluation de toutes les caractéristiques explicites de u n'est pas strictement nécessaire à la décision. De plus, pour $v \subset u$, on n'aura $\overline{\Phi}(v) - \overline{\Phi}(u \cap v) = \{1, 2, \ldots, n\} - \overline{\Phi}(u)$ que si u est le premier terme testé dans l'itération (2). Aussi, en général, cet algorithme évaluera beaucoup plus de caractéristiques que nécessaire.

Plusieurs auteurs se sont intéressés à cette approche (27,116,165) (dénommée "méthode des masques" dans la littérature sur les tables de décision). Une contribution typique est celle de King (116)qui, en vue de réduire le nombre de caractéristiques évaluées, propose quelques critères heuristiques pour ordonner les divers termes de l'itération (2). Cela conduit à un ordre d'évaluation des caractéristiques statiques, i.e., pratiquement indépendant de l'état e ; et surtout cela ne supprime pas les évaluations redondantes.

En fait, même en ordonnant aussi bien l'itération (2) que l'itération (2.1.), la méthode restera peu performante (relativement à l'évaluation des caractéristiques), et cela tant que le choix de la caractéristique testée à chaque étape ne tire pas profit au mieux de l'information acquise, i.e., de la valeur des caractéristiques précédemment évaluées sur e.

Cela nous conduit au deuxième type de processus décisionnels qui portent sur une structure arborescente obtenue par compilation du système de règles. Sauf mention explicite, on ne s'intéressera dans le reste de ce mémoire qu'à ce type de PDF.

Une partition π de $(\overset{n}{\underset{i=1}{X}} Z_i)$ est dite partition admissible de $S = \{X, A, D, R\}$ si π comporte (a + 1) blocs, chacun inclus dans l'un des ensembles $(R_1 \cup D)$, $(R_2 \cup D)$, ..., $(R_a \cup D)$, D ; seul le bloc inclus dans D pouvant être vide.

Un arbre de décision $t = \{A, X, V, L, \Gamma, \mathcal{L}, \mathcal{V}\}$ représente le système S si t réalise une partition admissible de S. Sur un tel arbre, une feuille correspondante à un r-cube inclus dans $(R_j \cup D)$ est associée à $\mathcal{L}(1) = A_j$. Une feuille incluse dans D est élaguée.

Notons que si t est faiblement équivalent à t', et si t représente S, alors t' aussi représente S.

Un processus décisionnel fermé sur un arbre t représentant S est défini par l'algorithme récursif suivant :

Procédure PDF (t, e) récursive :

1. Evaluer $X_i(e)$, avec $X_i = \mathcal{V}(v_1)$ caractéristique associée à la racine de t

2. Si la branche de v_1 correspondant à la valeur $X_i(e)$ mène à une feuille l, alors Retourner avec $A_j \leftarrow \mathcal{L}(l)$

3. Sinon faire :

3.1. t' ← sous-arbre de t engendré par $\hat{\Gamma}(v')$, avec v' fils de v_1 le long de la branche correspondante à $X_i(e)$

3.2. Appeler PDF (t', e)

4. Fin

Proposition 1.4. : Pour tout état $e \in E$, cet algorithme

- i) parcourt un chemin unique dans t de la racine à la feuille l qui correspond à la règle valide dans e ;

-ii) n'évalue le long de ce chemin une caractéristique X_i qu'au plus une fois.

Preuve : résulte immédiatement de la consistance et complétude de S et de la définition d'un arbre de décision représentant S.

°0°

I.4. PARAMETRES ET CRITERES D'OPTIMISATION

L'optimisation d'un processus décisionnel sur S consiste à rechercher un arbre t représentant S qui minimise la complexité de l'algorithme PDF selon un modèle réaliste du coût global de sa mise en oeuvre. Cette section présentera divers modèles et formulera les critères relativement auxquels l'optimisation sera entreprise.

Dans ce qui suit, on considèrera que les étapes (2), (3.1.) et (3.2.) de l'algorithme PDF ont un coût négligeable devant celui de l'étape 1 (évaluation d'une caractéristique). En effet, nous avons pu ramener ces étapes par un codage adéquat de l'arbre t à la lecture de deux entiers dans un tableau, à un test à zéro, et à un appel de programme ou un retour.

Comparativement à cela, chaque X_i est une procédure plus ou moins complexe et longue des données caractérisant le système, lesquelles données ne sont pas nécessairement présentes dans le processeur et doivent être acquises, éventuellement par des interactions très coûteuses avec l'environnement extérieur au processeur.

1.4.1. DISTRIBUTIONS DE PROBABILITE

Le calcul de la complexité moyenne de PDF nécessite la donnée d'une distribution de coût sur les caractéristiques, et d'une mesure de probabilité sur l'espace d'état E du système. L'hypothèse de la donnée d'une telle mesure de probabilité ne semble pas réaliste, surtout si l'on se préoccupe de trouver une procédure pour son acquisition dans une application concrète.

Une distribution de probabilité sur les sommets de l'espace de représentation ($\overset{n}{\underset{i=1}{X}} Z_i$) du système n'est pas plus facile à obtenir en l'absence d'hypothèse simplificatrice. Deux situations seront envisagées.

Dans un premier cas, on suppose connue pour chaque séquence d'actions A_j la probabilité d'avoir à exécuter cette séquence, ainsi que pour chaque caractéristique X_i une distribution de probabilité sur Z_i sachant A_j. L'estimation de ces $a*(1 + \Sigma z_i)$ paramètres scalaires ne pose pas de grandes difficultés si les caractéristiques sont statistiquement conditionnellement indépendantes. En l'absence de cette hypothèse, il faudrait estimer les distributions croisées de probabilités (Prob $\left[X_i(e) = k \mid A_j \text{ et } X_{j1}(e) = k_1, \text{ et } \ldots, \text{ et } X_{jr}(e) = k_r \right]$), ce qui est encore moins faisable que d'acquérir directement une distribution sur $(\overset{n}{\underset{i=1}{X}} Z_i)$.

Dans une deuxième approche, on remarque qu'un processus décisionnel ne permet pas d'observer individuellement chaque sommet élémentaire de $(\overset{n}{\underset{i=1}{X}} Z_i)$, puisque le processus s'arrête dès qu'une séquence d'actions est isolée. Seules les r - cubes correspondant aux feuilles de l'arbre de décision sont observables. On pourra acquérir, par estimation, une fréquence pour chacun d'entre eux et supposer, en l'absence d'informations supplémentaires, que les sommets d'un r-cube sont équiprobables. Ceci permettra de construire un autre arbre de décision sur lequel de nouvelles observations pourront être faites. La question importante de la convergence d'un tel processus ne sera pas abordée ici.

Dans l'un comme dans l'autre des deux cas précédents, on est en mesure de calculer, pour tout r-cube w la probabilité $\mu(w)$ que $(X_1(e), \ldots, X_n(e))$ soit l'un des sommets élémentaires de W.

Remarques :

- Pour $u \in D$, $\mu(u) = 0$;
- Sur un arbre t, $\mu(v)$ et $\mu(l)$ désignent les probabilités des termes associés au noeud v et à la feuille l respectivement, i.e., les probabilités dans un processus décisionnel sur t de passer par v et d'aboutir à l; on a :

$$\mu(v) = \sum_{v \in \Gamma^{-1}(\ell)} \mu(\ell) \qquad \text{(somme sur toutes les feuilles issues de v) ; et}$$

$$\sum_{L} \mu(\ell) = 1.$$

I.4.2. MODELE DES COUTS UNITAIRES

D'une manière générale, le coût d'évaluation d'une caractéristique X_i est une fonction, d'une part de l'état e du système sur lequel X_i est évaluée, et d'autre part de l'ensemble des caractéristiques précédemment évaluées. Il est peu vraisemblable que de telles fonctions puissent être déterminées, et même si elles le sont, leur évaluation sera aussi complexe que celle des caractéristiques dont elles donnent le coût. Aussi, est-il nécessaire de faire des hypothèses simplificatrices. Sur les sept modèles qui seront présentés (et dont seuls les deux premiers ont été abordés dans la littérature traitant de l'optimisation des arbres de décision), le plus simple est celui du modèle des coûts unitaires.

Selon ce modèle, toutes les caractéristiques ont le même coût constant, pris égal à 1. Le coût du PDF est alors égal à $|\Gamma^{-1}(1)|$ pour un appel menant à la feuille 1. La somme pondérée sur l'ensemble des feuilles de l'arbre donne la hauteur moyenne de t comme complexité du processus décisionnel.

L'optimisation des questionnaires(59,184) et des tables de décision (162,189)a été initialement étudiée sur ce modèle très simple auquel on ne s'intéressera plus dans ce qui suit.

I.4.3. MODELE DES COUTS CONSTANTS

On suppose la donnée de n paramètres $\{\rho_1, \ldots, \rho_n\}$ sur $\mathbb{R}^+$ ρ_i étant le coût d'évaluation de X_i, constant pour tout $e \in E$, et quelles que soient les autres caractéristiques préalablement évaluées. Ce modèle est assez réaliste si les caractéristiques du système ne font pas intervenir des traitements communs à plusieurs d'entr'elles, et si la dispersion du coût réel de chacune est assez faible autour du coût moyen.

Trois formulations de la complexité moyenne de PDF sur un arbre t peuvent être proposées dans ce modèle. La première est une somme pondérée du coût d'accès à chaque feuille de l'arbre :

$$\Psi(t) = \sum_{\ell \in L} \mu(\ell) \sum_{X_i \in \mathcal{V}(\hat{\Gamma}^{-1}(\ell))} p_i \qquad \text{(i)}$$

Exprimé matriciellement :

$$\Psi(t) = \bar{p} \cdot M \cdot \bar{\mu}_L \quad ; \quad \text{avec}$$

M = matrice binaire (q, n) dont l'élément (ligne j, colonne i) vaut 1 si X_i se trouve sur le chemin menant à la feuille l_j, i.e., si $X_i \in \mathcal{V}(\hat{\Gamma}^{-1}(\ell))$

$\bar{p}$ = vecteur $(p_1, \ldots, p_n)$ transposé ;

$\bar{\mu}_L$ = vecteur $(\mu(l_1), \ldots, \mu(l_q))$.

On peut mettre en relief le coût maximal du processus décisionnel :

$$\Psi(t) = \left(\sum_{i=1}^{n} p_i \right) - \sum_{\ell \in L} \mu(\ell) \sum_{X_i \notin \mathcal{V}(\hat{\Gamma}^{-1}(\ell))} p_i$$

Une deuxième formulation du coût exprimera le caractère récursif de l'algorithme PDF : son coût est la somme du coût d'évaluation de la caractéristique à la racine v_1 de l'arbre t, et des coûts des processus décisionnels sur chacun des sous-arbres issus de cette racine pondérés par leurs probabilités :

$$\Psi(t) = p(v_1) + \sum_{v \in \Gamma(v_1)} \mu(v) \cdot \Psi(t_v) \quad ; \quad \text{avec}$$

$$p(v_1) = p_i \quad \text{si } X_i = \mathcal{V}(v_1) \quad ; \quad \text{et}$$

t_v : sous-arbre engendré par $\hat{\Gamma}(v)$ pour $v \in \Gamma(v_1)$.

En étendant la récursion jusqu'aux feuilles de t, et en remarquant que sur une feuille l, $\Psi(t_l)$ est nul :

$$\Psi(t) = \sum_{v \in V} p(v) \, \mu(v) \qquad \text{(ii)}$$

Les formulations (i) et (ii) sont équivalentes.

(Preuve : $\Psi(t) = \sum \mu(\ell) \times \sum_{X_i \in \mathcal{V}(\hat{\Gamma}^{-1}(\ell))} \rho_i$ d'après (1) ; tous les noeuds de t apparaissent par l'intermédiaire de $\hat{\Gamma}^{-1}$ (1), et en factorisant la somme par noeuds, à $\rho(v)$ correspondra le facteur

$$\sum_{\ell \in \hat{\Gamma}(v)} \mu(\ell) = \mu(v) \qquad \square \ )$$

La troisième formulation est une somme sur les diverses caractéristiques qui résulte directement de la factorisation de (ii) :

$$\Psi(t) = \sum_{i=1}^{n} \rho_i \sum_{\mathcal{V}(v) = X_i} \mu(v) \qquad \text{(iii)}$$

On note qu'à une caractéristique X_i ne figurant dans aucun noeud de l'arbre, correspond un facteur $\sum_{\mathcal{V}(v) = X_i} \mu(v)$ nul.

Remarquons que si t et t' sont 2 arbres fortement équivalents, alors d'une part ils représentent un même système de règles (car faiblement équivalents), et d'autre part le PDF a le même coût sur ces deux arbres. En effet, t et t' définissent la même partition, donc à une permutation près, il leur correspond la même matrice M et le même vecteur $\bar{\mu}_L$.

I.4.4. MODELE DES COUTS DEPENDANTS

On se donne une partition de l'ensemble des caractéristiques $(\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_k)$ et k paramètres $\{\rho_1, \ldots, \rho_k\}$ sur $\mathbb{R}^+$. La classe de caractéristiques $\mathcal{C}_j$ traduit une relation de dépendance de coût entre ses membres. Si l'une quelconque des caractéristiques de $\mathcal{C}_j$ est évaluée, elle entraînera un coût constant ρ_j (coût de la classe) et fournira "gratuitement" la valeur des autres éléments de $\mathcal{C}_j$. Grâce à un tel modèle, on peut tenir compte des traitements communs, dans le cas par exemple d'un programme calculant simultanément plusieurs caractéristiques et qui ne peut être partitionné.

Le coût du processus décisionnel est directement obtenu à partir des formulations (i) ou (ii) de la section précédente. Par exemple :

$$\Psi(t) = \sum_{v \in V} \rho(v) . \mu(v) \quad \text{avec :}$$

$$\rho(v) = \begin{cases} \rho_k & \text{si } \mathcal{V}(v) \in \mathcal{C}_k \quad \text{et } \mathcal{V}(\hat{\Gamma}^{-1}(v) \cap \mathcal{C}_k = \emptyset \\ 0 & \text{sinon} \end{cases} ;$$

I.4.5. MODELE DES COUTS CONDITIONNELS

Pour généraliser le modèle précédent, on doit se donner le coût d'évaluation de chaque caractéristique X_i sachant les caractéristiques $X_{j1}, \ldots X_{jk}$ précédemment évaluées. Il n'est cependant pas très réaliste de supposer connu l'ensemble des $(n \times 2^{n-1})$ paramètres nécessaires à un tel modèle. On admettra donc que les dépendances de coût sont faibles, et on se contentera pour ce modèle d'un coût "à priori", plus quelques (2 à 3 en moyenne) coûts conditionnels par caractéristique.

Pour avoir $\Psi(t)$, on remplacera alors dans (ii) $\rho(v)$ par $\rho(v \mid \hat{\Gamma}^{-1}(v))$: coût conditionnel de $X_i = \mathcal{V}(v)$ sachant les caractéristiques évaluées avant X_i.

I.4.6. MODELE DES COUTS VARIABLES

Les coûts d'évaluation considérés dans les modèles précédents sont des moyennes sur l'ensemble des états $e \in E$. Dans certains cas, la dispersion des coûts réels est telle que ces moyennes sont peu significatives. Un modèle réaliste qui ne soit pas trop complexe supposerait ici que le coût d'évaluation de X_i est constant sur tous les états e pour lesquels X_i a la même valeur.

On se donne alors un ensemble de $(\sum_{i=1}^{n} z_i)$ paramètres sur $\mathbb{R}^+$:

$\rho(i,k)$: coût d'évaluation de X_i sur les états $\{ e \in E \mid X_i(e) = k \}$

Le coût d'un chemin de la racine d'un arbre t et une feuille l est : $\sum_{l(i) \neq \varphi} \rho(i, l(i))$; la somme pondérée sur toutes les feuilles donne :

$$\psi(t) = \sum_{\ell \in L} \mu(\ell) \times \sum_{\ell(i) \neq \varphi} \rho(i, \ell(i))$$

Pour retrouver une formulation telle que (ii), il suffit de définir $\rho(v)$ comme le coût moyen d'évaluation de $X_i = \vartheta(v)$ en ce noeud :

$$\rho(v) = \frac{1}{\mu(v)} \sum_{k=0}^{z_i - 1} \rho(i,k) \times \mu(\Gamma_k(v)) \; ; \quad \text{et}$$

$$\psi(t) = \sum_{v \in V} \sum_{k=0}^{z_i - 1} \rho(i,k) \times \mu(\Gamma_k(v))$$

On remarque que la même caractéristique X_i a des coûts moyens d'évaluation différents dans les divers noeuds de l'arbre où elle apparaît.

I.4.7. MODELE DES COUTS D'ACCES

L'évaluation d'une caractéristique correspond à l'exécution d'un code par un processeur, lequel devra d'abord accéder à ce code. Jusqu'à présent, on a supposé implicitement que toutes les caractéristiques étaient au même niveau d'accessibilité, et que leurs coûts d'évaluation intégraient un coût d'accès. Dans certaines applications, il est nécessaire de distinguer plusieurs supports de mémorisation d'un code avec des coûts d'accès et des espaces disponibles distincts.

On considèrera le cas d'un processeur dont la mémoire centrale ne permet de stocker qu'un sous-ensemble des caractéristiques du système, le reste étant mis en mémoire secondaire. Soit (I_1, I_2) une partition de $\{1,2, \ldots, n\}$ telle que :

- pour $i \in I_1$: X_i correspond à un logiciel mis en mémoire principale qui occupe l'espace ζ_i dans un espace total disponible γ ;

- pour $i \in I_2$: X_i est en mémoire secondaire, et son évaluation entraîne un coût d'accès ξ_i , en plus du coût d'évaluation.

On suppose que le coût d'accès pour la mémoire principale est nul, et que l'espace disponible en mémoire secondaire n'est pas contraint.

Le coût d'accès sur un arbre t et pour la partition (I_1, I_2) est alors:

$$\Psi_a(t) = \sum_{i \in I_2}^{n} \xi_i \times \sum_{X_i = v(v)} \mu(v) \quad ; \text{ en vérifiant la contrainte :}$$

$$\sum_{i \in I_1} \zeta_i \leq \gamma$$

Le coût total du PDF est :

$$\Psi(t) = \sum_{i=1}^{n} \rho_i \sum_{X_i = v(v)} \mu(v) + \sum_{i \in I_2} \xi_i \times \sum_{X_i = v(v)} \mu(v)$$

I.4.8. MODELE DES COUTS DE MAINTENANCE

L'ensemble des fonctions de coût considérées jusqu'à présent correspond à une complexité temporelle du PDF pour divers modèles du coût (en temps du processeur) de l'étape 1 de cet algorithme. Quelle serait une fonction de coût associée à sa complexité spatiale ?

La littérature traitant de l'optimisation des arbres de décision a abordé 2 types de critères : un correspondant au modèle des coûts constants (et la simplification dans le modèle des coûts unitaires), et un correspondant à l'espace mémoire occupé par un arbre(21,140,202,244) . La fonction optimisée est ici la somme sur tous les noeuds de la taille mémoire nécessaire pour stocker leurs caractéristiques.

En fait, un tel critère ne correspond à aucun modèle réaliste d'un processus décisionnel pour les raisons suivantes :

- le code associé à une caractéristique X_i n'est mémorisé qu'une seule fois, même si X_i figure dans plusieurs noeuds de l'arbre ;

- si on considère alors l'unique coût associé au codage de l'arborescence (on chercherait à minimiser $p = |V|$), ce coût devient négligeable relativement aux autres coûts du processus : l'arbre t peut être codé en une table de $(p + q)(1 + \max_i \{z_i\})$ entiers ;

- cette table peut, de plus, être réduite par la transformation de l'arbre en latticiel.

On admet donc que le nombre de noeuds d'un arbre t n'est pas un critère significatif. Par contre, chaque caractéristique X_i utilisée dans t peut entraîner un coût indépendant du nombre de noeuds où figure X_i et de son évaluation plus ou moins fréquente ; coût qui reflète entre autre l'espace occupé par X_i sur un support et le fait qu'elle soit accessible par t. Ce coût de maintenance de X_i dans t est, en toute rigueur, constant par unité de temps, mais on le ramène par commodité à un coût m_i constant par décision prise, en le supposant additif aux autres coûts du processus décisionnel. Si on ne considère que les coûts d'évaluation et de maintenance, on aura :

$$\Psi(t) = \sum_{v \in V} \rho(v)\, \mu(v) + \sum_{\exists v : X_i = \upsilon(v)} m_i$$

Combinés aux coûts d'accès, les coûts de maintenances peuvent être différenciés suivant que la caractéristique est mise en mémoire centrale ou en mémoire secondaire.

Plus généralement, on peut associer entre eux plusieurs des sept modèles de coûts présentés, les critères étant additifs.

Par exemple, les 4 modèles de coûts conditionnels, coûts d'accès, coûts variables et coûts de maintenance peuvent être combinés entre eux 2 à 2, 3 à 3, ou tous les 4,si on veut tenir compte à la fois de la valeur de X_i, des caractéristiques évaluées avant X_i, du coût d'accès à X_i à chaque fois que C_i est évaluée, et de son coût de maintenance qu'elle soit ou non évaluée.

Terminons cette section en mentionnant un dernier critère d'optimisation qui s'impose très naturellement lorsqu'on exprime le coût d'un PDF en terme de complexité algorithmique.Il s'agit de la complexité de l'algorithme compilant le système de règles en arbre de décision.

Le coût de génération d'un arbre t rejaillit bien évidemment sur le coût du PDF utilisant t, et ceci en fonction du nombre total d'utilisations de t. Si le système de règles n'est pas figé et s'il peut évoluer, soit au niveau de la logique même de règles, soit par simple dérive des paramètres qui y interviennent (coûts, probabilité), des recompilations seront nécessaires, et l'optimisation peut être formulée en terme d'une problématique intégrant à la fois la dynamique du système et les deux critères : coût de compilation d'un système de règles en arbre, coût du PDF sur cet arbre. Une approche à ce problème sera abordée au chapitre 4.

Les deux dernières sections de ce chapitre présentent des algorithmes exacts et approchés d'optimisation de PDF pour le seul modèle des coûts constants. Des généralisations seront développées au chapitre 3.

°0°

I.5. ALGORITHMES APPROCHES D'OPTIMISATION

On présente, dans un premier temps, un algorithme non déterministe de génération d'arbres de décision représentant un système S. Cet algorithme sera considéré comme élément générique d'une classe d'algorithmes approchés d'optimisation obtenus en fixant le choix non déterministe par diverses méthodes et fonctions développées par la suite.

I.5.1. UN ALGORITHME NON DETERMINISTE

En vue de construire un arbre t représentant un système S, introduisons les notions de sous-arbre partiel et de sous-système de règles associé à un terme. Un sous-arbre partiel τ est soit un arbre vide τ_o (i.e., V = L = Ø), soit un arbre de décision τ dont au moins un chemin issu de la racine ne se termine pas par une feuille. La dernière branche sur un tel chemin est dite branche pendante. A cette branche est associé, comme pour tout chemin sur un arbre de décision, un terme de $(\overset{n}{\underset{i=1}{\times}} Z_i)$. Si $X_{i_1}, \ldots, X_{i_k}$ sont les caractéristiques des noeuds rencontrés sur ce chemin et si $j_1, \ldots, j_k$ sont les indices sur $Z_{i_1}, \ldots, Z_{i_k}$ des branches du chemin, alors le terme qui lui est associé est celui représenté par le monôme :

$$[(X_{i_1}(e) = j_1) \wedge \ldots \wedge (X_{i_k}(e) = j_k)].$$

Par convention τ_o à une seule branche pendante, il lui correspond l'hypercube $(\overset{n}{\underset{i=1}{\times}} Z_i)$ entier.

Soit u un terme quelconque. Le sous-système de règles de S associé à u est par définition :

$$S/u = \{X/u,\ A/u,\ D\ ,\ R/u\} \quad ; \quad \text{avec :}$$

$X/u = \{X_i \in X \mid i \in \Phi(u)\}$: caractéristiques non spécifiées dans u,

$R/u = \{R_j \in R \mid R_j \cap u \not\subset D\}$;

$A/u = \{A_j \in A \mid R_j \in R/u\}$.

L'algorithme suivant permet de générer un arbre de décision représentant un système de règles S :

Algorithme A1

Données d'entrée : un système de règles de décision $S = \{X, A, D, R\}$.

1. Initialisation : $\tau \leftarrow \tau_o$ arbre partiel vide ;

2. Itérer tant que (τ n'est pas un arbre complet)

2.1. Prendre une branche pendante dans τ , soit u le terme correspondant ;

2.2. Si R/u comporte une règle de décision unique R_j, Alors augmenter τ le long de la branche pendante par une feuille l avec $\mathcal{L}(l) = A_j$;

2.3. Sinon Faire ;

2.3.1. Choisir dans X/u une caractéristique X_i ;

2.3.2. Augmenter τ le long de la branche pendante par un noeud v, avec $\mathcal{V}(v) = X_i$;

2.3.3. Associer à ce noeud z_i branches pendantes indicées par $\{0, 1, \ldots z_i - 1\}$;

2.3.4. Elaguer toute branche issue de v qui correspond à un terme $\subset D$;

2.4. Fin de l'itération 2.

3. Fin ;

Données de sortie : l'arbre de décision $t \leftarrow \tau$.

Proposition 1.5. : L'arbre t généré par A1 représente S.

Preuve : S étant consistant et complet par hypothèse, $\forall u \in U$ ou bien $u \in D$ et la branche correspondante a été élaguée (2.3.4.), ou bien $R/u \neq \emptyset$. Dans ce dernier cas, si R/u comporte plus d'une règle, alors nécessairement $X/u \neq \emptyset$ et l'algorithme peut se poursuivre jusqu'à l'obtention d'un arbre complet. Il reste à établir que 1) t est un arbre de décision valide, et 2) qu'il définit une partition admissible pour S. La première proposition résulte du fait que la caractéristique X_i associée à un noeud v est choisie dans X/u (2.3.2.), et donc que X_i ne figure dans aucun des noeuds de $\hat{\Gamma}^{-1}(v)$: tout chemin de t rencontre une caractéristique au plus une fois. Pour justifier la deuxième proposition, on remarque que t définit une partition π de $(\overset{n}{\underset{i=1}{X}} Z_i)$ dont tout bloc, autre que ceux de D (branches élaguées), correspond à un terme u tel que $R/u = \{R_j\}$ règle unique : u est inclus dans $(R_j \cup D)$. Donc, π est plus fine qu'une partition admissible pour S. □

La complexité de A1 est proportionnelle au nombre de branches de t, i.e., en O (p + q) ; p = |V|, q = |L|. Exprimée en fonction des paramètres d'entrée, cette complexité est dans le pire des cas en $O(\prod_{i=1}^{n} z_i)$ si l'arbre t a toutes ses feuilles de rang n.

On remarque que 2 étapes de cet algorithme font intervenir un choix non spécifié. Celui de l'étape (2.1.) n'intervient que dans l'ordre d'exploration de l'itération 2, i.e., l'ordre de développement de l'arbre partiel. La fonction de choix dans l'étape (2.3.1.) fait, par contre, intervenir un non-déterminisme vrai.

Calculons le nombre maximum d'arbres représentant un système S, i.e., susceptibles d'être générés par A1 du fait de son non-déterminisme. Pour un système de n caractéristiques, à la première itération de A1, n choix sont possibles en (2.3.1.), chacun ajoutant z_j branches pendantes, et donc faisant intervenir z_j choix non-déterministes sur des sous-systèmes de (n - 1) caractéristiques :

$$\Xi(n) = n \quad \Xi^{z_j}(n-1) \; ; \text{ et } \quad \Xi(1) = 1 \quad ; \quad \text{d'où :}$$

$$\Xi(n) = n \quad (n-1)^{z_{j1}} \quad (n-2)^{z_{j1}\, z_{j2}} \ldots (n-k)^{z_{j1}\cdots z_{jk}} \ldots 2^{z_{j1}\cdots z_{j\,n-2}} \; ; \qquad (i)$$

$(j_1, \ldots, j_n)$ étant une permutation de (1, ..., n) pour laquelle $(z_{j1}, \ldots, z_{jn})$ est une séquence non croissante.

$\Xi(n)$, majorant du nombre d'arbres distincts représentant S, n'est atteint que si S comporte $a = \prod_{i=1}^{n} z_i$ séquences d'actions (et donc, D = ∅).

Plusieurs simplifications peuvent être faites sur l'arbre t résultant de l'algorithme A1 sans altérer les propriétés de PDF relativement à S. Du fait de l'élagage des branches correspondantes à des dépendances, certains noeuds peuvent ne plus avoir qu'une seule branche. La caractéristique testée dans un tel noeud ne contribue en rien au processus décisionnel : son unique valeur possible est prévisible dès l'évaluation précédente. Une telle caractéristique est redondante. Pour éviter son apparition dans un arbre t, il suffit de modifier la définition de X/u comme suit :

$$X/u = \left\{ X_i \in X \;\middle|\; i \in \Phi(u), \text{ et } \exists\, j,\ k \neq j \text{ tel que } (u/i \leftarrow j) \notin D \text{ et } (u/i \leftarrow k) \notin D \right\}.$$

Une telle définition garantit l'existence pour chaque noeud d'au moins 2 branches j et k ne correspondant pas à des dépendances.

Dans une deuxième simplification de t, si 2 branches d'indices j et k issues d'un noeud v mènent à deux sous-arbres identiques, l'un d'eux peut être supprimé et les deux branches regroupées en une seule indexée par le couple (j,k). A l'issue d'une telle simplification $\Gamma(v)$ ne définira plus une partition de Z_i, domaine de $X_i = \nu(v)$, en singletons, mais en blocs quelconques. Il en résulte que la partition de $(\bigtimes_{i=1}^{n} Z_i)$ définie par t ne sera plus une partition homogène.

La simplification précédente peut être généralisée : si deux sous-arbres de t engendrés par $\hat{\Gamma}(v)$ et $\hat{\Gamma}(v')$ sont identiques, alors l'un d'eux peut être supprimé. On transforme, dans ce cas, la branche $(\Gamma^{-1}(v'), v')$ en $(\Gamma^{-1}(v'), v)$, si le sous-arbre de racine v' est celui qui est supprimé. A l'issue d'une telle transformation, t n'est plus une arborescence, mais un latticiel qui a exactement q = a feuilles, une par séquence d'actions A_j.

L'algorithme A1 peut être modifié, de façon à générer directement t sous la forme simplifiée finale de latticiel. Néanmoins, on ne retiendra pour l'instant que la première des 3 simplifications, car elle seule influe sur la complexité de PDF.

En effet, les deux autres transformations réduisent le nombre de noeuds de t, mais ne modifient en rien la composition de ses différents chemins et les probabilités de leurs parcours.

I.5.2. DIVERSES FONCTIONS DE CHOIX HEURISTIQUES

Un "oracle" pour un algorithme non déterministe est une procédure qui, parmi diverses alternatives, désigne toujours le meilleur choix relativement à l'objectif qui est poursuivi. Dans notre cas, il s'agit d'optimiser l'arbre t pour le critère Ψ sur les distributions de coût et de probabilité ρ et μ. L'oracle dont on aurait besoin remplacerait l'étape (2.3.1.) de A1. Appelé sur l'argument u, il fournirait en retour la caractéristique $X_i \in X/u$ qui assure l'optimalité de t.

A défaut de proposer un oracle infaillible, la littérature concernée par l'optimisation des arbres de décision est extrêmement riche en bons conseils et recettes permettant de faire un choix plus ou moins judicieux d'une caractéristique X_i. Il s'agit toujours de définir un ordre total sur X/u par l'intermédiaire d'une fonction $Y : X/u \rightarrow \mathbb{R}$, dite fonction d'évaluation ou de choix heuristique, et de prendre dans l'étape (2.3.1.) de A1 l'élément extrémal relativement à cet ordre.

En résumant assez schématiquement une littérature abondante, on distinguera 5 principaux types de fonctions heuristiques. Les heuristiques du 1er et 2ème types sont basées sur l'idée que plus le recouvrement que définit une caractéristique X_i sur R se rapproche d'une partition fine et équilibrée (i.e., à parts égales), plus i est efficace ; et qu'au contraire, si toutes les règles se retrouvent dans chacun des termes du recouvrement, l'évaluation de X_i aura peu contribué à isoler une décision unique.

Pour des modèles simplifiés de tables de décision, Egler (39) , puis Press (196) proposent de choisir dans X/u la caractéristique X_i correspondant à un nombre minimal de termes ayant "φ" pour $i^{ème}$ coordonnée.

L'algorithme de Pollack (189) procédé également à ce comptage de coo-données "φ", mais y introduit une pondération confuse qui tient compte à la fois des probabilités et des cardinalités (nombres de sommets élémentaires équivalents) des termes. Yasui (244), puis Verhelst (239)améliorent cette pondération. Ils font intervenir les coûts (constants) des caractéristiques, et procèdent, préalablement au comptage, à des regroupements de termes au sein d'une règle en vue de maximiser le nombre de "φ", ce qui permet de traduire réellement les recouvrements dans une caractéristique.

Après un tel regroupement, la fonction d'évaluation heuristique est :

$$Y_1(X_i) = \rho \sum_{u(i)=\varphi} \mu(u) \qquad ; \text{ on choisit } X_i \text{ à } Y_1 \text{ minimale.}$$

L'idée d'une partition équilibrée apparaît chez Montalbano (162) ("delayed-rule method") pour les tables de décision, et chez de nombreux auteurs (59,172) pour les procédures d'identification(Binary Splitting Algorithm). Dans la formulation de cette heuristique due à Lemaitre(138)on commence par maximiser les regroupements des termes dans chaque règle et relativement à chaque caractéristique, puis on détermine (dans le cas binaire) :

$$Y_2(X_i) = \frac{1}{\rho_i} \cdot \min \left\{ \sum_{u(i)=1} \mu(u) \; ; \sum_{u(i)=0} \mu(u) \right\} ;$$

On choisit la caractéristique ayant Y_2 <u>maximale</u>. Cette heuristique se généralise au cas non binaire.

Le troisième type de fonction heuristique est basé sur la théorie de l'information. Schwayder(222)développe une définition de l'entropie d'une caractéristique et propose de choisir X_i dont la fonction entropie serait maximale. Ganapathy et Rajaraman(56), puis Schwayder(223)améliorent cette heuristique : après maximisation des regroupements à l'intérieur de chaque règle relativement à une caractéristique X_i :

$$Y_3(X_i) = \frac{1}{\rho_i} \sum p_j \operatorname{Log} \frac{1}{p_j} \qquad ; \text{ avec } P_j = \sum_{u(i)=j} \mu(u)$$

On choisit X_i à $Y_3(X_i)$ <u>maximale</u>.

Ce type de critère reste très populaire en optimisation des questionnaires. Picard(184,185)et d'autres chercheurs(33,35,84) ont étudié très extensivement les propriétés de la fonction d'information de Shannon et d'autres types de fonctions d'information, appliquées à ce problème. Néanmoins, selon l'expérimentation de Lemaitre(138)sur les questionnaires réalisables, l'heuristique Y_2 donnerait des résultats nettement meilleurs (en proximité par rapport à l'optimum) que Y_3.

Le quatrième type de fonction heuristique donne un contenu plus rigoureux à l'approche de Yasui et de Verhelst. Cette approche est basée sur le calcul de la probabilité d'atteindre une règle de décision sans évaluer une caractéristique X_i. La fonction d'évaluation, proposée dans (67) en tant que généralisation de celle développée par Reinwald et Soland (201)et utilisée parallèlement par d'autres chercheurs(149,164), est :

$$Y_4 (X_i/u) = \rho_i \times q_i (u) \qquad \text{avec}$$

$q_i(u)$ = Probabilité d'atteindre dans S/u une règle de décision sans évaluer X_i.
Le calcul de $q_i(u)$ sera développé ultérieurement et les performances de Y_4 seront analysées dans le chapitre 4.

A partir d'une fonction heuristique quelconque Y, on peut construire une autre fonction heuristique Y' de la façon suivante :

$$Y'(X_i/u) = Y(X_i/u) + \sum_{0 \leq j < z_i} \min_{X/(u/i \leftarrow j)} \left\{ Y(X_k/(u/i \leftarrow j)) \right\}$$

$(u/i \leftarrow j)$ étant obtenu à partir de u en fixant la $i^{ème}$ coordonnée à la valeur j, et $Y(X_k/u) = 0$ si S/u contient une règle unique.

Cette heuristique au deuxième ordre assure, en général, de meilleurs résultats que Y, mais elle est plus complexe à calculer : si le calcul de Y est en $O(f(n))$, alors celui de Y' est en $O(f(n) + z_i . f(n-1))$. Avec quelques variantes, Verhelst(239)(en présentant à tort son algorithme comme une optimisation exacte), ainsi que Sethi et Chatterjee(221)ont préconisé une telle approche heuristique sur la base de Y_1.

Il est possible de procéder à l'extension d'une heuristique Y au-delà du deuxième ordre. L'algorithme devient alors du type Programmation Dynamique à horizon limité (approche proposée dans un cas particulier par Slagle et Lee(226).

Soit $A_1(Y)$ l'algorithme approché d'optimisation obtenu en fixant le choix non déterministe de A_1 par la fonction heuristique Y. Les performances de cet algorithme peuvent être jugées sur la base de sa complexité et de la proximité à l'optimum des solutions qu'il fournit.

Concernant la complexité de $A_1(Y)$, (une analyse détaillée est reportée au chapitre 4), on peut remarquer que :

1) $A_1(Y)$ ajoute à la complexité de A_1 celle de calcul et de la minimisation (ou maximisation) de Y sur X/u, ainsi que la complexité de l'étape initiale de prétraitement de S pour certaines heuristiques (maximisation des " φ " sur chaque caractéristique) ;

2) Pour aucune heuristique connue, $A_1(Y)$ n'a été prouvé polynomial. Ceci est du au fait que cet algorithme peut générer un arbre de taille exponentielle en la dimension de S (sauf si S est restreint au modèle des procédures d'identifications, car un questionnaire a toujours $a = |\mathcal{A}|$ feuilles, et donc un nombre de noeuds $p \leq a$) ;

3) Les quatre heuristiques Y_1, Y_2, Y_3, Y_4 présentées ici sont équivalentes, au sens de la complexité de l'algorithme $A_1(Y_i)$ résultant.

Il ne sera pas plus simple de juger et de comparer les divers algorithmes $A_1(Y_i)$ sur la base de leur degré d'approximation. En effet, pour aucune heuristique Y, on ne connaît de majorant constant, ni de caractérisation en moyenne, de l'écart relatif ou absolu par rapport à l'optimum des solutions fournies par $A_1(Y)$. Pour presque toutes les fonctions Y, des exemples ont été trouvés, montrant que $A_1(Y)$ pouvait générer des solutions arbitrairement éloignées de l'optimum

(Yasui(244)en propose pour Y_1, Moret(164)pour Y_4 et l'exemple de Garey et Graham (59) couvre Y_2 et Y_3). De plus, la littérature ne présente que quelques rares résultats, très peu significatifs, de temps de calcul sur des exemples académiques ; et les études statistiques comparatives sur la base de données générées aléatoirement sont pratiquement inexistantes. A ma connaissance, les exceptions sont :

- Lemaitre(138)qui a pû conclure à la supériorité de Y_2 sur Y_3 pour l'optimisation des questionnaires (2 points de mesure : 10 et 15 caractéristiques binaires, chacun moyenné sur 100 essais) ;

- Schumacher et Sevcik(218)qui ont testé A1(Y_3) sur des tables de décision, les solutions obtenues s'écartaient de l'optimum de 2 % en moyenne, et au plus de 9 % (2 points de mesure : 5 et 8 caractéristiques, nombre d'essais non précisé).

°0°

I.6. ALGORITHMES EXACTS D'OPTIMISATION

Comme pour de nombreux problèmes d'optimisation combinatoires difficiles, les seuls algorithmes exacts d'optimisation des PDF connus sont des algorithmes énumératifs et reposent soit sur une exploration ascendante par composition de sous-problèmes du type Programmation Dynamique, soit sur une recherche descendante dans une arborescence, du type PSES (recherche en profondeur ou "backtrack search") ou PSEP (recherche ordonnée, "Best Search", ou "Branch-and-Bound").

I.6.1. PROGRAMMATION DYNAMIQUE

Dans le modèle des coûts constants, on a précédemment obtenu une formulation récursive du coût $\Psi(t)$ d'un arbre t de racine v_1 :

$$\Psi(t) = \rho(v_1) + \sum_{v \in \Gamma(v_1)} \mu(v)\ \Psi(t_v) \qquad \text{(i)}$$

$\varphi(v_1)$: coût de la caractéristique X_i tel que $X_i = \upsilon(v_1)$;

$\mu(v)$: probabilité du terme correspondant au chemin menant à v ;

t_v : sous-arbre de t engendré par $\hat{\Gamma}(v)$.

Définissons pour un terme quelconque $u \in U$, la fonction f(u) par :

- si u tel que R/u comporte une règle unique ou bien est vide, alors f(u) = 0 ;
- sinon $f(u) = \mu(u)\ \Psi(\hat{t}_u)$; $\hat{t}_u$ sous-arbre optimal représentant S/u.

L'expression précédente (i) de Ψ donne alors pour ce dernier cas :

$$f(u) = \min_{i \in \Phi(u)} \left\{ \mu(u)\ \rho(X_i) + \sum_{0 \leq j < z_i} f(u/i \leftarrow j) \right\} \qquad \text{(ii)}$$

Le problème de la recherche d'un arbre minimal relativement au critère Ψ se ramène alors au problème de la résolution de l'équation récurrente(ii) pour le n-cube u_o. La formulation de cette résolution en programmation dynamique est immédiate :

Algorithme A2

Données d'entrée : Système de règles S consistant et complet, distributions ρ et μ ;

1. Itérer sur k = 1, 2, ..., n ;

1.1. Itérer sur tous les k-cubes $u \in U$,

1.1.1. Si R/u est vide ou se ramène à une règle unique, Alors f(u) ← 0

1.1.2. Sinon $f(u) \leftarrow \min_{i \in \Phi(u)} \{ \mu(u)\ \rho(X_i) + \sum_{0 \le j < z_i} f(u/i \leftarrow j) \}$

1.1.3. Fin de l'itération 1.1.

1.2. Fin

Données de sortie : $f(u_o)$ coût d'un arbre optimal $\hat{t}$ représentant S, $\hat{t}$ pouvant être généré récursivement à partir de u_o en remplaçant un k-cube par un noeud contenant la caractéristique X_i correspondante au minimum de (1.1.2.), ou par une feuille si f(u) = 0 (feuille éventuellement élaguée).

La preuve de cet algorithme est bien connue dans sa formulation générale (13) . La complexité de A2 est en première approximation en $O(\prod_{i=1}^{n} (1+z_i))$; i.e. proportionnelle au nombre total de termes dans U. Dans une analyse plus fine, on doit tenir compte pour chaque k-cube, $1 \le k \le n$, de k comparaisons dans l'étape (1.1.2.), car $|\Phi(u)| = k$. Le nombre de k-cubes est égal à la somme de C^n_{n-k} produits d'ordre (n-k) de termes distincts de $\{z_1, \ldots, z_n\}$, i.e. :

$$\sum_{1 \le j_1 \le k+1} \ \sum_{j_1 < j_2 < k+2} \ \cdots \ \sum_{j_{n-k-1} < j_{n-k} \le n} z_{j_1} z_{j_2} \cdots z_{j_{n-k}} \quad ; \text{(iii)}$$

Les 0-cubes n'étant pas explorés (car, par définition, f(u) = 0 pour un 0-cube), le nombre total de comparaisons dans A_2 est :

$$\sum_{i \leq k \leq n} k \left[\sum_{i \leq j_1 < k+1} \; \sum_{j_1 < j_2 < k+2} \cdots \sum_{j_{n-k-1} < j_{n-k} \leq n} \beta_{j_1} \beta_{j_2} \cdots \beta_{j_{n-k}} \right]$$

Cette expression se ramène à (preuve par récurrence) :

$$\sum_{i \leq k \leq n} \; \prod_{\substack{i \leq j \leq n \\ j \neq k}} (1 + \beta_j) \qquad \text{(iv)}$$

Remarquons que du fait de l'exploration ascendante systématique de la Programmation Dynamique, la complexité moyenne de cet algorithme est du même ordre que sa complexité au pire des cas : l'itération (1.1) est toujours effectuée $\left[\prod_{1 \leq i \leq n}(1+\beta_i) - \prod_{1 \leq i \leq n} \beta_i\right]$ fois, seules les k comparaisons de la minimisation de (1.1.2.) sont évitées dans certains termes.

Compte-tenu du fait que les valeurs de f(u) trouvées à l'itération (1) sur k ne doivent être retenues que jusqu'à l'itération suivante sur (k + 1), la complexité spatiale de A2 est proportionnelle à la valeur maximale de l'expression (iii). Dans le cas de caractéristiques binaires, cette expression se ramène à $\left[C^{n}_{n-k}\, 2^{n-k}\right]$ dont le maximum est atteint pour $k = \frac{n}{3}$.

Dubail(33) et Garey (57,58) ont étudié l'application de la Programmation Dynamique à l'optimisation des questionnaires. Un algorithme similaire à A2 fut proposé par Schumacher et Sevcik(218)sur un modèle restreint de tables de décision. Expérimentant sur des tables comportant jusqu'à 10 caractéristiques binaires, ces auteurs ont constaté une croissance de la complexité moyenne de leur algorithme, aussi bien spatiale que temporelle, en $O(3^n)$. Ils en affirment, néanmoins, la supériorité sur l'approche "Branch-and-Bound" de(201) (infaisable au-delà de 5 caractéristiques selon eux), et le faible écart en complexité par rapport à un algorithme approché tel que celui de (165) dont la croissance moyenne serait en $O(n^2.2^n)$. Lew(140)a généralisé l'algorithme de Schumacher et Sevcik à un modèle plus large de tables de décision. Tout en en vantant les mérites, il admet néanmoins que l'utilisation de la Programmation Dynamique est limitée à des tables d'une douzaine de caractéristiques binaires.

I.6.2. PROCEDURES DE SEPARATION ET D'EVALUATION, RECHERCHE ARBORESCENTE

Ce type de procédure d'optimisation repose sur le principe de la division de l'espace initial des solutions du problème en sous-espaces de plus en plus réduits. La décomposition (on dit aussi "séparation") se fait en général par des partitions successives, ce qui confère à la recherche une structure arborescente. La procédure développe et explore cette arborescence selon une certaine stratégie jusqu'à ce qu'un sous-ensemble singleton contenant une solution optimale soit isolé .

L'arborescence sur l'ensemble des solutions du problème d'optimisation des PDF est introduite simplement par une relation d'ordre sur l'ensemble $T = \{ t_1, t_2, \ldots, t_{\Xi(n)} \}$ des arbres de décision représentant un système S.

Les noeuds $\{ v_1, \ldots, v_p \}$ d'un arbre t sont indicés suivant l'ordre scriptural (préordre des rangs, et pour les noeuds ayant même rang, ordre d'alignement transverse gauche-droite). On associe à t la liste ordonnée correspondante $(\mathcal{V}(v_1), \ldots, \mathcal{V}(v_p))$ de ses caractéristiques. L'ordre lexicographique (à partir des indices sur X) permet de comparer les listes associées à deux arbres t et t', et d'induire sur T une relation d'ordre total.

Deux arbres t et t' sont dits k-équivalents si leurs k premiers noeuds respectifs sont 2 à 2 identiques. La partition induite sur T par cette relation d'équivalence est désignée par $P_k = \{ \tau_{k1}, \ldots, \tau_{k\alpha} \}$.

Chaque classe d'équivalence τ_k est un sous-arbre partiel constitué par les k premiers noeuds communs aux arbres de cette classe. On note :

$\mathcal{T} = \bigcup_{k=0}^{n} P_k = \{ \tau_0, \tau_1, \ldots \}$ l'ensemble des sous-arbres partiels, avec

$P_0 = \{ \tau_0 \}$; τ_0 : sous-arbre vide, ou classe d'équivalence contenant tout T ;

$P_M = T$: partition dont chaque classe contient un arbre complet unique (les arbres de T n'ayant pas tous le même nombre de noeuds, M correspondra au maximum de ce nombre, en convenant qu'une partition sur une classe réduite à un arbre unique ne modifie pas cette classe).

$\mathcal{T}$ possède une structure d'arborescence dont τ_0 est la racine, les sous-arbres partiels de P_k en sont les éléments de rang k, et si $\tau \in P_k$, $\tau' \in P_{k+1}$ avec $\tau' \subset \tau$, alors τ' est successeur immédiat de τ .

L'algorithme non déterministe A1 parcourt un et un seul chemin dans l'arborescence $\mathcal{T}$, de la racine τ_o à une feuille t représentant un arbre complet. Le passage d'un élément τ de rang k à son successeur $\tau' \subset \tau$ se faisant par le choix de la caractéristique X_i à affecter au $(k+1)^{ème}$ noeud (au sens de l'ordre scriptural) de τ. Comme on l'a déjà mentionné, aucune méthode connue ne permet de faire les choix qui mèneraient, en fin de parcours, à une solution optimale, ou même à une solution dont l'écart relatif à l'optimum est borné à priori.

Les Procédures de Séparation et d'Evaluation apportent un ensemble de techniques d'exploration de $\mathcal{T}$, en vue de visiter un nombre limité d'arbres partiels, tout en isolant en fin de recherche une solution optimale. On ne reprend pas ici un exposé détaillé de ces méthodes qui sont relativement bien connues (Cf. Lawler-Wood(137), Mitten(161), ou Roy (209) par exemple), mais on en développe brièvement quelques applications à l'optimisation des processus décisionnels.

Les procédures d'exploration par Séparation et Evaluation Séquentielle (PSES, "depth-first", ou "backtrack search") sont basées sur un développement en profondeur de $\mathcal{T}$. En chaque noeud τ de l'arborescence :

- on calcule le coût de τ en supposant qu'il s'agit d'un arbre complet ($\Psi(\tau) = \sum_{v \in \tau} \rho(v)\ \mu(v)$) ;
- on ordonne les caractéristiques X_i à affecter au $(k+1)^{ème}$ sommet de τ par l'intermédiaire d'une des fonctions heuristiques Y_i ;
- on poursuit sur le successeur τ' de τ utilisant la première caractéristique relativement à Y_i.

Arrivé à une feuille t de $\mathcal{T}$, on reprend l'exploration en profondeur à partir du premier ascendant τ de t ayant des successeurs τ' non encore explorés. De plus, si $\hat{t}$ est le meilleur arbre complet atteint à un certain moment, l'algorithme abandonnera l'exploration du noeud τ (et de toute la sous-arborescence de $\mathcal{T}$ qui en est issue) si $\Psi(\tau) \geq \Psi(\hat{t})$.

Lemaitre(38)a proposé, pour l'optimisation des questionnaires réalisables, un algorithme reprenant un principe d'exploration équivalent à celui-ci, mais dans une formulation récursive.

Transposé dans notre cas, cet algorithme serait le suivant :

Algorithme (récursif) A3 (u)

1. Initialisation : $\Psi^* \leftarrow \infty$, $\tau \leftarrow \tau_o$

2. Itérer sur $\{ X_i \in X/u \}$

2.1. Affecter à la racine de τ la caractéristique i

2.2. $\Psi(\tau) \leftarrow p_i \cdot \mu(u)$

2.3. Itérer sur $\{ j \mid 0 \leq j \leq z_i - 1 \}$ tant que
$v \leftarrow (u/i \leftarrow j)$
Si $v \subset D$, élaguer la $j^{ème}$ branche issue de X_i

2.3.3. Sinon Si R/v comporte une règle unique R_k, alors augmenter τ le long de la $j^{ème}$ branche de X_i par une feuille l avec $\mathcal{L}(l) = A_j$.

2.3.3.2. Sinon faire :
$t \leftarrow$ A3 (v)
Augmenter τ le long de la $j^{ème}$ branche de X_i par le sous-arbre complet t
$\Psi(\tau) \leftarrow \Psi(\tau) + \Psi(t)$

2.3.4. Fin itération 2.4.

2.4. Si $\Psi(\tau) \leq \Psi^*$ alors $\tau^* \leftarrow \tau$ et $\Psi^* \leftarrow \Psi$

2.5. Fin itération 2

3. Retourner τ^*

4. Fin.

Cet algorithme (improprement dénommé PSES) est en fait un mélange d'exploration en profondeur (succession d'appels récursifs jusqu'à l'obtention d'un terme réduit à une règle unique) et de recherche en largeur (itération (2) sur toutes les caractéristiques de X/u, et retour au niveau appelant qu'après avoir trouvé un sous-arbre complet optimal correspondant à u). Un des avantages essentiels de la PSES, qui est d'atteindre très rapidement une solution complète lors de la première descente en profondeur, et d'améliorer successivement cette relation dans la suite de l'exploration, est perdu par l'algorithme A3, car il n'atteint un arbre complet qu'en isolant l'arbre optimal (retour de l'appel sur $u_o = (\varphi, \ldots, \varphi)$). Par contre, cet algorithme est plus simple à implémenter qu'une procédure PSES (Cf. détails et résultats expérimentaux de temps de calcul dans(138)).

Les procédures de Séparation et Evaluation Progressive (PSEP, "Best-Search", "Branch-and-Bound") reposent essentiellement sur une fonction d'évaluation $f(\tau)$ qui est un estimateur du coût $\Psi(\hat{t})$ de l'arbre complet optimal inclus dans la classe d'équivalence τ. L'arborescence $\mathcal{T}$ est construite au fur et à mesure à partir de τ_o par "développement" d'arbres partiels. Lorsque le noeud τ est développé, on génère et ajoute à $\mathcal{T}$ tous ses successeurs $\tau' \subset \tau$ en déterminant pour chacun d'eux son évaluation $f(\tau')$. Le noeud développé à chaque itération est celui qui parmi tous les noeuds non développés de $\mathcal{T}$ possède l'évaluation minimale. Si ce noeud est une feuille de $\mathcal{T}$ (i.e., un arbre complet), la procédure s'arrête. On démontre que si l'évaluation f est :

- monotone, i.e., $\forall\ \tau'$ successeur de τ : $f(\tau) \leq f(\tau')$; et
- coïncidente, i.e., $\forall t \in \tau$, t arbre complet : $f(t) = \Psi(t)$,

alors le noeud t provoquant l'arrêt de l'algorithme correspond à un arbre complet optimal.

Une PSEP est plus difficile à implémenter et à mettre en oeuvre qu'une PSES. De plus, elle nécessite l'élaboration d'une fonction d'évaluation monotone et coïncidente, et le calcul (parfois long) de la valeur de cette fonction sur chaque noeud visité de l'arborescence de recherche $\mathcal{T}$.

Par contre, la PSEP a deux avantages déterminants :

- elle génère une solution optimale en visitant, en général, beaucoup moins de noeuds de $\mathcal{T}$ que ne le ferait une PSES ;

- elle permet d'atteindre une solution sous-optimale à écart relatif à l'optimum majoré par un paramètre ε donné à priori, et dans ce cas, la recherche dans $\mathcal{T}$ est encore plus rapide.

Reinwald et Soland(201,202) ont proposé un algorithme du type PSEP pour la traduction optimale de tables de décision entièrement développées (contenant 2^n règles pour n caractéristiques binaires). Il est possible de développer entièrement toute table de décision (en remplaçant chaque r-cube par ses 2^r sommets élémentaires), mais le développement augmente considérablement la dimension de la table. Cela restreint l'utilisation de cet algorithme et justifie les critiques qu'il a rencontrés (Cf., par exemple (56,221) ou (218) pour qui cet algorithme nécessiterait "plusieurs heures de calcul pour n = 6 (caractéristiques) et des années et des siècles de calcul au-delà de n = 6"). A ma connaissance, cet algorithme ne fut pas implémenté (Reinwald et Soland ont préféré développer une PSES utilisant la même fonction d'évaluation que(201)).

On présente dans (67) un algorithme de même type basé sur une fonction d'évaluation généralisant celle de Reinwald et Soland.

Dans la version de recherche d'un arbre ε-optimal,i.e., dont l'écart relatif à l'optimum est majoré par ϵ , cet algorithme est le suivant.

Algorithme A 4

Données d'entrée : $S = \{X, A, D, R\}$, distributions p et μ, fonction f, paramètre $\varepsilon \geq 0$

1. Initialisation : $\tau \leftarrow \tau_0$; $P \leftarrow \{\tau_0\}$

2. Itérer tant que (τ n'est pas un arbre complet)

2.1. Supprimer τ de P

2.2. Soit u le terme correspondant à la première branche pendante de τ (au sens de l'ordre scriptural, i.e., si τ est de rang k dans $\mathcal{T}$, c'est la branche qui mènerait au $(k+1)^{ème}$ noeud)

2.3. Si $(u \subset D)$, alors élaguer cette branche pendante de τ

2.4. Sinon Si (R/u comporte une règle unique R_k), alors augmenter τ le long de la $j^{ème}$ branche de X_i par une feuille l avec $\mathcal{L}(l) = A_k$

2.4.2. Sinon faire :

$F \leftarrow \infty$

2.4.2.2. Itérer sur $\{X_i \in X/u\}$

Soit τ' le successeur de τ obtenu en augmentant la branche u par un noeud contenant X_i

Evaluer $f(\tau)$; mettre τ' dans P

Si $(F > f(\tau'))$, alors faire : $\tau_1 \leftarrow \tau'$; $F \leftarrow f(\tau')$

Fin itération 2.4.2.2.

2.4.2.3. Soit $\hat{\tau}$ l'arbre de P ayant l'évaluation f minimale

2.4.2.4. Si $[(f(\tau_1) - f(\hat{\tau}))/f(\hat{\tau}) \leq \varepsilon]$, alors $\tau \leftarrow \tau_1$

2.4.2.5. Sinon $\tau \leftarrow \hat{\tau}$

2.5. Fin itération 2

3. Fin

Données de sortie : L'arbre de décision $t \leftarrow \tau$ de coût $\Psi(t) \leftarrow f(\tau)$

L'arbre t est ε-optimal, en particulier pour $\varepsilon = 0$, il est exactement optimal. Les définitions de la fonction f pour les divers critères d'optimisation introduits et les preuves de ses propriétés seront présentées au Chapitre 3.

Comme tout autre algorithme du type PSES ou du type PSEP, A4 procèdera, dans le pire cas, à autant d'itérations qu'il y a d'arbres partiels dans $\mathcal{T}$. Sa complexité est donc en $O(|\mathcal{T}|)$. Si les n caractéristiques de X sont toutes binaires, on vérifie que :

$$|\mathcal{T}| = 1 + n + \sum_{k=0}^{n-3} \sum_{j=1}^{2^{k+1}} \left[\prod_{i=0}^{k} (n-i)^{2^i} \right] (n-k-1)^j$$

La table suivante donne quelques valeurs de cette expression :

n	1	2	3	4	5	6	7	8	9	10	11	12
$\lvert\mathcal{T}\rvert$	2	7	22	1133	$3,31.10^{6}$	$3,03.10^{13}$	$3,82.10^{27}$	$5,83.10^{55}$	$1,53.10^{112}$	$1,17.10^{225}$	$7,51.10^{450}$	$3,36.10^{902}$

L'expression littérale de $|\mathcal{T}|$ est trop complexe dans le cas de caractéristiques quelconques. Sachant que $\mathcal{T}$ possède $|T| = \Xi(n)$ feuilles, on peut prendre comme approximation :

$$|\mathcal{T}| = 2\ \Xi(n) = 2 \times n \cdot \prod_{k=1}^{n-2} (n-k)^{\prod_{j=1}^{k} z_{ij}}$$

En pratique, l'évaluation f est souvent suffisamment discriminante pour permettre d'utiliser A4 sur des systèmes ayant jusqu'à 10 caractéristiques binaires avec $\varepsilon = 0$, et au-delà pour des valeurs plus grandes de ε. On trouvera dans [67] des détails d'implémentation et quelques résultats sur les performances de cet algorithme.

°0°

I.7. CONCLUSION

Résumons rapidement les performances des algorithmes d'optimisation présentés dans ce chapitre :

- l'algorithme A1(Y) fournit une approximation dont l'écart à l'optimum ne peut être borné, et sa complexité est en $O\left(\prod_{i=1}^{n} z_i\right)$
- la programmation dynamique (A2) est en $O\left(\prod_{i=1}^{n} 1 + z_i\right)$;
- les procédures de recherche arborescente (A3 et A4) sont en $O\left(n \prod_{k=1}^{n-2} (n-k)^{\prod_{j=1}^{k} z_j}\right)$.

Les raisons des comportements non polynomiaux de ces algorithmes (y compris de celui non déterministe A1) seront établis au Chapitre 4. Mais, dès à présent, on voit la nécessité d'un algorithme fournissant une approximation garantie et dont la complexité soit bien inférieure à celle de A4. Un tel algorithme sera établi comme cas particulier d'une procédure générale de recherche heuristique dans des hypergraphes. Le Chapitre suivant est consacré à cette famille de procédures.

°0°

CHAPITRE 2

ALGORITHMES QUASI-ADMISSIBLES DE RECHERCHE HEURISTIQUE DANS DANS LES GRAPHES ET LES HYPERGRAPHES

2.1. INTRODUCTION

2.2. RECHERCHE HEURISTIQUE ADMISSIBLE DANS UN GRAPHE : A^*

Convergence et admissibilité d'A^*
Complexité d'A^*
Monotonie de l'heuristique

2.3. RECHERCHE HEURISTIQUE ϵ-ADMISSIBLE DANS UN GRAPHE : A_ϵ

Heuristique monotone
Heuristique α-minorante
Complexité d'A_ϵ

2.4. RECHERCHE HEURISTIQUE DANS UN HYPERGRAPHE : AO^*

2.5. RECHERCHE HEURISTIQUE DANS UN HYPERGRAPHE : AO_ϵ

Complexité de la recherche heuristique dans un hypergraphe
Stratégie persévérante pour AO_ϵ
Généralisation à des hypergraphes orientés quelconques

2.6. RECHERCHE HEURISTIQUE ET OPTIMISATION COMBINATOIRE

2.7. CONCLUSION

°0°

2.1. INTRODUCTION

On s'intéresse, ici, aux problèmes de recherche de chemins dans un graphe, ou de sous-arbres dans un hypergraphe, optimaux pour certains critères additifs. Au-delà de la résolution de ces problèmes, les algorithmes développés sont destinés, en fait, à optimiser des processus décisionnels, et cela à deux niveaux :

- au niveau des processus fermés pour déterminer un arbre de décision représentant un système de règles et optimisant la recherche de la ou des règles valides dans un état donné ;

- au niveau des processus ouverts pour optimiser (au sens de sa complexité) la recherche d'une séquence de règles qui permettra d'amener le système d'un état initial vers un état terminal.

Sans détailler ici ce dernier problème, on admettra qu'il se pose par la donnée d'un ensemble (fini) de règles de décision portant sur un système décrit dans un espace d'état discret fini ou infini. Chaque règle est un opérateur de changement d'état, et une ou plusieurs règles sont valides dans un état donné. On dispose également d'un état initial et d'un ensemble d'états terminaux. Une solution au problème est une séquence de règles (un "plan") qui amènera le système de l'état initial vers un état terminal.

Pour ce type de problème, le terme de recherche heuristique désigne, d'une manière imprécise, toute procédure qui utilise pour atteindre une solution, une fonction évaluant heuristiquement (sur la base d'informations données à priori, et d'une manière approximative) l'écart en coût entre un état donné et un état terminal.

L'essentiel du chapitre porte sur deux algorithmes de recherche heuristique originaux A_ε et AO_ε, qui généralisent respectivement deux algorithmes connus A^* et AO^*.

Ces deux derniers algorithmes sont admissibles, en ce sens que moyennant certaines propriétés de la fonction heuristique utilisée, ils fournissent une solution garantie optimale. Les arguments qui justifient la nécessité et l'intérêt des algorithmes proposés sont les suivants :

- A^* et AO^* ne considèrent, comme critère d'optimisation, que le coût d'une solution (i.e., le coût de l'application du plan généré), et ne sont pas en mesure de prendre en compte le coût de recherche de cette solution ;

- ce dernier coût est, pour de nombreuses applications, du même ordre de grandeur que celui optimisé, et traduit en fait la complexité très élevée d'A^* et d'AO^*, et

- à cause de cette complexité, dans de nombreux cas, la recherche admissible est impraticable.

A_ε et AO_ε sont des algorithmes de recherche ϵ- admissibles, i.e., des algorithmes qui fournissent une solution à écart relatif à l'optimum majoré par ε, paramètre fourni à priori. Chacun d'entre eux constitue en fait, plutôt qu'un algorithme unique, une classe de procédures ("schéma d'approximation", selon la terminologie de(62)) dans laquelle l'utilisateur, en fixant la valeur du paramètre ϵ (ainsi qu'en optant pour certaines stratégies d'exploration), choisira un algorithme particulier réalisant un compromis entre les deux critères mentionnés.

Les algorithmes A^* et AO^* sont présentés avec des améliorations de détail et quelques propriétés originales. Certains résultats les concernant, et sur lesquels des erreurs ou des confusions ont été relevées dans la littérature sont corrigés ; d'autres résultats sont entièrement redémontrés, quand on a pensé pouvoir améliorer les preuves précédemment publiées.

Les algorithmes A_ε et AO_ε sont longuement discutés et développés, chacun à la suite de l'algorithme qu'il généralise. Leurs preuves de convergence et d' ε- admissibilité, et certaines de leurs propriétés sont établies. Leur complexité n'est analysée qu'asymptotiquement.

La dernière section est consacrée à quelques parallèles entre les procédures de recherche heuristique et les méthodes d'optimisation combinatoire par séparation et évaluation (PSE). Les algorithmes A^* et AO^* ont été développés pour résoudre des problèmes d'Intelligence Artificielle, mais en ignorant presque totalement les résultats acquis en Recherche Opérationnelle sur des problèmes similaires. La littérature ne mentionne que quelques remarques sur d'éventuelles ressemblances qui transparaissent, malgré le parti pris d'opter pour des notations et une terminologie complètement distinctes entre les deux disciplines (Cf. par exemple Pohl (188) ou Ibaraki(101)).

On montre ici que, dans quelques cas particuliers, les algorithmes A^* et "Branch-and-Bound" sont identiques, et que certaines extensions du "Branch-and-Bound" rejoignent A^* et AO^* dans le cas général.

°□°

2.2. RECHERCHE HEURISTIQUE ADMISSIBLE DANS UN GRAPHE : A^*

Soit S un ensemble de règles de décision portant sur un système décrit dans un espace d'état discret U. On modélise ce système par un graphe orienté[1] $G = (U, \Gamma)$ dont chaque sommet $u \in U$ est un état, et v est fils de u ($v \in \Gamma(u)$) s'il existe dans S une règle de décision valide dans l'état u, qui, si elle est appliquée, amène le système à l'état v. Le graphe G est valué pour refléter la structure de coût du système : on associe à chaque arc (u,v) le coût, $K(u,v) \geq 0$, d'application de la règle de décision correspondante. On suppose les coûts additifs.

Le système étant dans un état initial u_0, on désire l'amener dans un état terminal quelconque dans l'ensemble $W \subset U$; en minimisant le coût total de la transformation. Il suffit, pour cela, de trouver dans G un chemin (u_0-w) de coût minimal, pour $w \in W$.

En reprenant les notations de (170), on définit :

- $K^*(u,u')$: coût d'un chemin minimal de u à u', s'il existe au moins un chemin entre ces deux sommets. Sinon, $K^*(u,u')$ n'est pas défini ;

- $g^*(u) = K^*(u_0,u)$: coût minimal d'accès de l'état initial à l'état u ;

- $h^*(u) = \min_{w \in W} \{K^*(u,W)\}$: coût d'un chemin minimal de u à un état terminal ;

- $f^*(u) = g^*(u) + h^*(u)$: coût d'un chemin minimal (u_0-w) contraint à passer par l'état u ; $f^*(u_0)$ est donc le coût d'un chemin optimal.

Ces fonctions n'étant en général pas connues, une procédure de recherche de chemin dans G utilisera leurs estimateurs :

- g(u) : estimateur de $g^*(u)$. On prend le coût du meilleur chemin (u_0-u) connu au moment de cette estimation. On a alors : $g(u) \geq g^*(u)$;

---

(1) L'ensemble de règles est connu explicitement, mais le graphe d'état est implicite.

- $h(u)$: estimateur de $h^*(u)$, représente l'information heuristique disponible à priori sur le problème ;

- $f(u) = g(u) + h(u)$: estimateur de $f^*(u)$.

Rappelons quelques propriétés de l'estimateur h. On dit que h est une heuristique

- presque parfaite ("Non- misleading heuristic", Ibaraki (99)) ssi : $\forall u, v \in U$ tels que $h^*(u)$ et $h^*(v)$ soient définis : $h(u) < h(v) \Rightarrow h^*(u) < h^*(v)$

- consistante : $\forall u, v \in U$ tels que $K^*(u,v)$ soit défini : $h(u) \leq h(v) + K^*(u,v)$

- monotone : $\forall u, v \in \Gamma(u)$: $h(u) \leq h(v) + K(u,v)$

- localement consistante : $\forall u$ tel que $\Gamma(u) \neq \emptyset$, $\exists v \in \Gamma(u)$: $h(u) \geq h(v)$

- minorante : $\forall u$ tel que $h^*(u)$ soit défini : $h(u) \leq h^*(u)$

- coïncidente : $\forall w \in W$: $h(w) = 0$

Ces propriétés sont données par ordre de restrictivité non croissante. La dernière suppose tout simplement la faculté de reconnaître qu'un sommet est terminal, ce qui sera admis par la suite. Comme on le verra, la monotonie pourra être établie pour de nombreux estimateurs. Par contre, à ma connaissance, aucune heuristique presque parfaite n'a été développée sur un problème réel, bien que la recherche guidée par une telle heuristique soit de complexité minimale (99) .

L'algorithme suivant est une synthèse de nombreuses contributions (66, 146,188) faites à un algorithme initialement proposé par Hart, Nilson et Raphaël (82,83) [1]. Chaque itération de l'algorithme développe un sommet de G, i.e. génère ses successeurs.

---

(1) Pour certains auteurs, l'algorithme s'appelle A^* ou A, suivant que h est ou n'est pas minorante(170,171).. De nombreuses autres dénominations du même algorithme, avec quelques variantes, sont utilisées :
HS, B, HPA, R, Q ... [146,163,188) .
On s'en tiendra à la plus répandue (A^*), et cela indépendamment des propriétés de h.

Les sommets générés sont partitionnés en deux ensembles :

Q : ensemble des sommets développés ;

P : ensemble des sommets pendants, i.e., susceptibles d'être développés.

L'ensemble P est totalement ordonné : par la fonction f dans le sens croissant, et pour les sommets ayant même estimateur f, par la fonction g dans le sens décroissant, avec priorité aux sommets terminaux.

De plus, on note :

$\hat{u}$: premier sommet de P au sens de l'ordre précédent ;

γ^{-1} : pointeur vers le père unique d'un sommet, correspondant au meilleur chemin de la racine à ce sommet trouvé à une certaine étape :

y : paramètre scalaire désignant $\max\{ f(u) \mid u \in Q \}$ à l'étape courante.

Algorithme A^*

Données d'entrée : graphe $G = (U, \Gamma)$, sommet initial u_o, sommets terminaux $W \subset U$, valuations K positives ou nulles sur les arcs de G, fonction h sur U

1. Initialisation : $P \leftarrow \{u_o\}$; $Q \leftarrow \emptyset$; $g(u_o) \leftarrow 0$; $f(u_o) \leftarrow h(u_o)$; $y \leftarrow 0$;

2. Itérer tant que ($P \neq \emptyset$ et $\hat{u} \notin W$)

(choix du sommet u à développer)

2.1. Si $f(\hat{u}) \geq y$, alors prendre $u \leftarrow \hat{u}$;

2.2. Sinon prendre u tel que : g(u) est minimal, avec $u \in P$ et $f(u) < y$

(développement du sommet u)

2.3. Supprimer u de P et le mettre dans Q ; $y \leftarrow \max\{ y; f(u) \}$;

2.4. Itérer sur $\{v \in \Gamma(u) \mid (v \notin P \text{ et } v \notin Q) \text{ ou } g(v) > g(u) + K(u,v)\}$:

2.4.1. $g(v) \leftarrow g(u) + K(u,v)$;

2.4.2. $f(v) \leftarrow g(v) + h(v)$;

2.4.3. $\gamma^{-1}(v) \leftarrow u$;

2.4.4. Ranger v dans P suivant l'ordre f croissant, g décroissant ;

2.4.5. Fin de l'itération 2.4.

2.5. Fin de l'itération 2.

3. Fin.

Données de sortie : Si ($\hat{u} \in W$) : le chemin $(u_o - \hat{u})$ obtenu en traçant les pointeurs γ^{-1} à partir de $\hat{u}$; sinon ($P = \emptyset$), il n'y a pas de chemin de u_o à un sommet de W.

A^* procède à une recherche guidée par l'estimateur f. Si on ne tient pas compte pour le moment du test (2.1.), destiné à réduire la complexité et sur lequel on reviendra ultérieurement, l'ordre défini sur P conduit à développer à chaque itération le sommet estimé sur le meilleur chemin menant à une solution (f minimal), et parmi diverses alternatives équivalentes, à préférer le sommet estimé le plus proche en coût d'un sommet terminal (g maximal).
Remarquons que d'autres stratégies de recherche pourraient être utilisées si on modifie l'ordre sur P : ainsi, en rangeant (étape 2.4.3.) les sommets dans l'ordre FIFO ("First-In-First-Out"), on obtiendrait une exploration en largeur, l'ordre LIFO ("Last-In-First-Out") conduirait à une recherche en profondeur.

Exemple 2.1. : Soit G le graphe valué suivant avec $W = \{u_9, u_{12}\}$
et $h(u_0) = 19$, $h(u_1) = 16$ ..., etc. (Figure 2.1.)

La séquence d'itérations d'A^* sur G est donnée par le tableau suivant :

Sommet u développé	f (u)	Successeurs générés ou modifiés	Successeurs non modifiés	Ensemble ordonné P	y
é	0+19=19	u_1, u_2, u_3, u_4	∅	$u_3(5+14)$; $u_1(3+16)$; $u_2(4+16)$; $u_4(6+15)$	19
u_3	5+14=19	u_6	∅	u_1 ; u_2 ; u_4 ; $u_6(14+9)$	19
u_1	3+16=19	∅	u_2, u_4	u_2 ; u_4 ; u_6	19
u_2	4+16=20	∅	u_3	u_4 ; u_6	20
u_4	6+15=21	u_5, u_6	u_3	$u_5(13+9)$; u_6 (13+19)	21
u_5	13+9=22	u_7, u_8, u_9	∅	u_6 ; $u_8(18+6)$; $u_7(17+7)$; $u_9(25+0)$	22
u_6	13+9=22	u_7, u_{10}, u_{11}	∅	$u_{10}(19+3)$; $u_7(15+7)$; u_{11} (15+8) ; u_8 ; u_9	22
u_{10}	19+3=22	u_{12}	u_{11}	u_7 ; $u_{12}(23+0)$; u_{11} ; u_8 ; u_9	22
u_7	15+7=22	u_9	u_3, u_{12}	u_{12} ; u_{11} ; $u_9(24+0)$, u_8	22
u_{12}	23+0=23	$u_{12} \in W$ ⇒ arrêt d'A^* et sortie du chemin ($u_0, u_4, u_6, u_{10}, u_{12}$)			

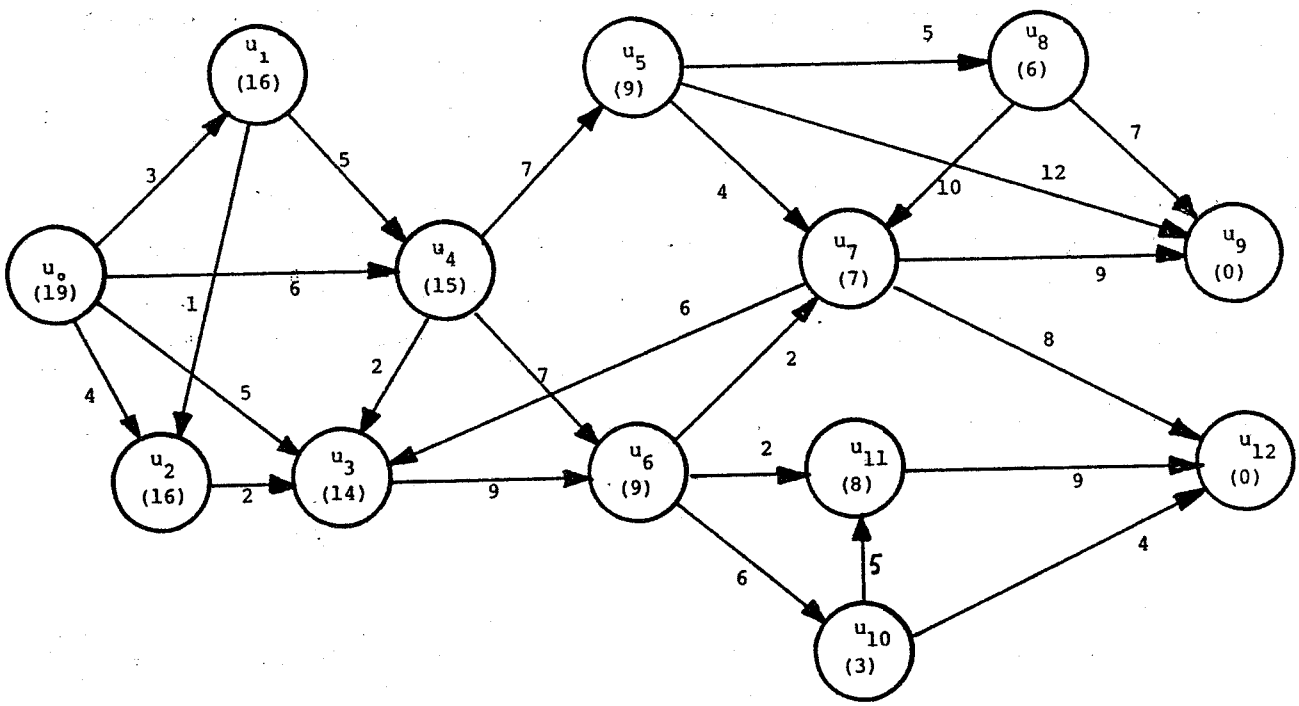


FIGURE 2.1.

Passons en revue les principales propriétés de l'algorithme :

Convergence et admissibilité d'A^* : Nilsson(170,171) établit la convergence et l'admissibilité d'A^* pour les graphes à valuations strictement positives. En fait, cette dernière contrainte est restrictive sans être nécessaire : comme le remarque Vanderburg (237), les propriétés de l'algorithme restent valides sur des δ - graphes à valuations positives ou nulles. Tout graphe qui n'admet pas de chemin longueur infinie et de coût fini est un δ-graphe.

Proposition 2.1. : Si G est un graphe fini à valuation positive ou nulle, alors A^* s'arrête soit sur un sommet terminal, soit faute de sommet pendant (P = Ø) s'il n'existe pas de chemin de u_o à un sommet terminal.

Preuve : Un sommet développé ($\in$ Q) ne peut redevenir pendant ($\in$ P) que si on trouve un nouveau chemin qui y mène strictement meilleur que celui précédemment connu (amélioration de g(v) en 2.4.1.). Eventuellement, tous les chemins menant à un sommet sont explorés, et du fait des coûts positifs ou nuls, un cycle ne peut être parcouru plus d'une fois. □

L'admissibilité de l'algorithme sera établie en trois étapes :

Proposition 2.2. : Si h est un estimateur minorant, à toute itération d'A^*, tout chemin optimal menant à un sommet terminal possède au moins un sommet pendant u_i tel que $f(u_i) \leq f^*(u_o)$.

Preuve : Considérons la séquence de sommets u_o, u_1, ..., $u_l = w$ constituant un chemin optimal menant à un sommet terminal. Pour toute itération d'A^*, l'un au moins des sommets de cette séquence est pendant : u_o ayant été développé, son fils u_1 est soit pendant, soit développé ; et, dans ce dernier cas, u_2 est également soit dans P, soit dans Q,... etc. Si u_i est le premier sommet de cette séquence à être pendant, le chemin optimal (u_o, u_i) ayant été exploré par A^*, on a : $g(u_i) = g^*(u_i)$. Si h est un minorant $h(u_i) \leq h^*(u_i)$, d'où : $f(u_i) = g(u_i) + h(u_i) \leq f^*(u_i) = f^*(u_o)$, car u_i est sur un chemin optimal menant à W. □

Proposition 2.3. : Si h est un estimateur minorant, tout sommet u développé par A^* est tel que $f(u) \leq f^*(u_o)$.

Preuve (par récurrence) : Cela est vrai à la première itération, car on développe u_o, et $f(u_o) = h(u_o) \leq h^*(u_o) = f^*(u_o)$ du fait de l'hypothèse sur h.

Si le résultat est vrai jusqu'à l'itération j, alors à ce moment $\forall u \in Q$, $f(u) \leq f^*(u_o)$ et comme $y = \max\{f(u) \mid u \in Q\}$, on a $y \leq f^*(u_o)$.

A l'itération (j + 1), deux cas sont possibles :

1) A^* choisit de développer $\hat{u}$ (étape 2.1.) et, dans ce cas, $f(\hat{u}) = \min \{ f(u) \mid u \in P \}$ par définition. D'après le lemme précédent, il existe dans P un sommet u_i tel que $f(u_i) \leqslant f^*(u_o)$, or $f(\hat{u}) \leqslant f(u_i)$, ce qui établit le résultat;

2) A^* choisit de développer (étape 2.2.) un sommet u, tel que $f(u) \leqslant y$, or $y \leqslant f^*(u_o)$. □

Proposition 2.4. : Si G est un δ- graphe fini ou infini à valuations positives ou nulles dans lequel existe au moins un chemin (u_o, w), $w \in W$, et si h est un estimateur minorant, alors A^* est admissible : il s'arrête en fournissant un chemin de coût minimal menant à un sommet terminal.

Preuve : L'argument sur les graphes finis concernant les sommets qui quittent définitivement P, parce que tous les chemins qui y mènent ont éventuellement été explorés, s'applique de nouveau ici. Il faut, de plus, établir que P ne croît pas indéfiniment avec de nouveaux sommets. Soit G' le sous-graphe de G engendré par tous les sommets u atteignables à partir de u_o par un chemin de coût $g^*(u) \leqslant f^*(u_o)$. G étant un δ- graphe, G' est nécessairement fini. Tout sommet u n'appartenant pas à G' est tel que: $f(u) \geqslant g(u) \geqslant g^*(u) > f^*(u_o)$: A^* ne peut développer un tel sommet d'après la proposition précédente. Seuls les sommets du sous-graphe fini G' sont éventuellement explorés et, parmi eux, A^* finit nécessairement par rencontrer un sommet terminal : $\hat{u} \in W$. Par définition, $f(\hat{u}) = \min \{ f(u) \mid u \in P \} \leqslant f^*(u_o)$ et $f^*(u_o) = \min \{ f^*(w) \mid w \in W \} \leqslant f^*(\hat{u}) \leqslant f(\hat{u})$, car u est terminal ; d'où $f(\hat{u}) = f^*(u_o)$. □

Complexité de l'algorithme A^* : A partir de l'analyse faite dans [105] sur l'algorithme de Dijkstra (recherche sans information heuristique de plus courts chemins dans un graphe), Martelli[146] a pû établir que l'algorithme A^* dans la formulation de Hart, Nilsson et Raphaël[82] (qui développe systématiquement à chaque itération le sommet $\hat{u}$) avait une complexité exponentielle en $O(2^N)$, N étant le nombre de sommets du graphe ; et que moyennant l'adjonction d'un test sur le sommet à développer (étapes 2.1. et 2.2.), la complexité de l'algorithme était toujours plus faible et se ramenait, dans le pire des cas, en $O(N^2)$.

Exposons schématiquement les grandes lignes de son argumentation. Une heuristique h, même minorante, peut conduire à développer un même sommet autant de fois qu'il y a de chemins qui y mènent. Le nombre total d'itérations est alors proportionnel au nombre de chemins menant à tous les sommets du graphe, d'où la complexité exponentielle. Partant du fait que pour tout sommet u développé par l'algorithme : $f(u) \leqslant f^*(u_o)$ (Proposition 2.3.), on en déduit que le coût d'un chemin optimal (u_o, w) ne saurait être inférieur à $y = \max \{ f(u) \mid u \in Q \}$. En conséquence, tous les sommets $u \in P$ tel que $f(u) \leqslant y$ devront être développés. Pour éviter qu'ils ne le soient de nombreuses fois, Martelli propose, dans ce cas, d'ignorer l'information heuristique et de les explorer dans l'ordre de g croissant, qui est la stratégie de l'algorithme de Dijkstra. On récupère alors la complexité en $O(N^2)$ de cet algorithme qui n'examine un arc du graphe qu'au plus une fois. Remarquons, enfin, que l'analyse de Martelli reste valable pour des δ - graphes infinis : N désignera le nombre de sommets du sous-graphe G' (plus exactement, dans [146] N est le nombre de sommet $u \in U$ tel que $g^*(u) + h(u) \leqslant f^*(u_o)$).

Dans un cas particulier important, l'algorithme A^* possède d'intéressantes propriétés ; il est en particulier de complexité linéaire en O(N). Examinons ce cas.

Monotonie de l'heuristique : La littérature n'est pas très claire sur les propriétés de l'heuristique h et, en particulier, celles à la base des propositions qui vont suivre. Initialement, Hart, Nilsson et Raphaël(82,83) établissent ces résultats en supposant h consistante et minorante. Pohl(188) affirme que la consistance est une propriété trop forte et restrictive et qu'une heuristique monotone et minorante suffit. Nilsson[171] corrige son texte en reprenant à son compte cette proposition.

Or, on peut y relever les inexactitudes suivantes :

- l'hypothèse d'une heuristique minorante est simplement superflue ;

- monotonie et consistance sont des propriétés équivalentes ;

- la définition que Pohl et Nilsson proposent pour la monotonie ($\forall u, v \in \Gamma(u)$: $h(v) \leq h(u) \leq h(v) + K(u,v)$) est beaucoup plus contraignante qu'il n'est nécessaire. En fait, leur définition rend la monotonie plus restrictive que la consistance.

Etablissons les relations entre consistance, monotonie et minoration.

Proposition 2.5. : Si h est monotone et coïncidente, alors elle est aussi minorante.

Preuve (par l'absurde) : Si h n'est pas minorante $\exists\, u \in U$ tel que $h(u) > h^*(u)$: $h^*(u)$ étant défini, il existe une séquence $(u, u_1, \ldots, u_1, w)$ constituant un chemin optimal (u,w) :

$$\left.\begin{array}{l} h(u_1) - h(w) \leq K(u_1, w) \\ \vdots \\ h(u_1) - h(u_2) \leq K(u_1, u_2) \\ h(u) - h(u_1) \leq K(u, u_1) \end{array}\right\} \Rightarrow h(u) - h(w) \leq K(u, u_1) + \ldots + K(u_1, w)$$

Or, $h(w) = 0$; et $h^*(u) = K(u, u_1) + \ldots + K(u_1, w)$ □

Proposition 2.6. : h est monotone ssi h est consistante.

Preuve (directe) : Soit u et v tel que $K^*(u,v)$ est défini ; notons $(u, u_1, \ldots, u_1, v)$ un chemin optimal de u à v :
$K^*(u,v) = K(u, u_1) + \ldots + K(u_{1-1}, u_1) + K(u_1, v)$.

Si h est monotone, alors :

$$\left.\begin{array}{l} h(u) \leq h(u_1) + K(u, u_1) \\ h(u_1) \leq h(u_2) + K(u_1, u_2) \\ \vdots \\ h(u_1) \leq h(v) + K(u_1, v) \end{array}\right\} \quad \text{d'où } h(u) \leq h(v) + K^*(u,v) \text{ ; et h est consistante.}$$

(Réciproque) : Soit u et $v \in \Gamma(u)$. Par définition, $K^*(u,v) \leq K(u,v)$
si h est consistante, $h(u) \leq h(v) + K^*(u,v) \leq h(v) + K(u,v)$ et h est monotone. □

Par ailleurs, h est bien évidemment coïncidente si elle est minorante (car $\forall u$: $0 \leqslant h(u)$; et $0 \leqslant h(w) \leqslant h^*(w) = 0$). Cependant dans de nombreux cas, on utilisera avantageusement les propriétés locales de coïncidence et de monotonie pour établir la minoration.

Revenons donc aux propriétés d'A^* pour une heuristique monotone.

Proposition 2.7. : Si h est monotone et coïncidente, alors f est non décroissante sur la séquence de sommets développés par A^*.

Preuve (par récurrence) : A la deuxième itération, tous les sommets de P sont fils de u_o ; pour chacun : $f(u) = K(u_o,u) + h(u) \geqslant h(u_o) = f(u_o)$ par monotonie. Donc, la base de la récurrence est établie. Supposons la proposition vraie jusqu'à l'itération j : f étant non décroissante sur la séquence de sommets développés, le sommet u_j développé à cette itération est tel que $f(u_j) \geqslant f(u)$ $\forall u \in Q$, d'où $f(u_j) \geqslant y = \max\{ f(u) \mid u \in Q \}$. Donc, u_j a été sélectionné (étape 2.1.) comme premier sommet de P. Soit u_{j+1} le sommet développé à l'itération suivante. Si u_{j+1} se trouvait dans P à l'itération j, alors d'après ce qui précède, $f(u_j) \leqslant f(u_{j+1})$, et f serait non décroissante. Sinon u_{j+1} serait un successeur de u_j et, dans ce cas, u_{j+1} n'aurait été mis dans P qu'en affectant (étape 2.4.1.)
$g(u_{j+1}) \leftarrow g(u_j) + K(u_j, u_{j+1})$, d'où :

$$f(u_{j+1}) = g(u_{j+1}) + h(u_{j+1}) = g(u_j) + K(u_j, u_{j+1}) + h(u_{j+1}) \geqslant g(u_j) + h(u_j) = f(u_j). \quad \square$$

On en déduit immédiatement que si h est monotone, à chaque itération le sommet $\hat{u}$ est tel que $f(\hat{u}) \geqslant \max\{ f(u) \mid u \in Q \} = y$. Donc A^* développera systématiquement le premier sommet de P, et les étapes (2.1.) et (2.2.) pourront être supprimées dans cette hypothèse.

Proposition 2.8. : Si h est monotone et coïncidente, alors pour tout sommet u développé par A^*, on a : $g(u) = g^*(u)$, i.e., u ne peut être développé qu'au plus une fois, et le chemin $(u_o - u)$ trouvé par l'algorithme est optimal.

Preuve (par récurrence) Base : Trivialement vrai à la première itération $g(u_o) = g^*(u_o) = 0$.

Induction : Soit u_{j+1} le sommet développé à l'itération (j+1). La proposition étant supposée vraie jusqu'à l'itération j, u_{j+1} est développé pour la première fois. De plus, d'après la proposition précédente à l'itération (j+1) : $f(u_{j+1}) = \min_{u \in P} f(u)$. Soit la séquence $(u_o, u_1, \ldots, u_l, u_{j+1})$ constituant un chemin optimal (u_o, u_{j+1}). Si aucun sommet de $(u_o, u_1, \ldots, u_l)$ ne figure dans P à l'itération (j+1), c'est qu'ils ont tous été développés, et donc qu'un chemin optimal menant à u_{j+1} est connu : $g(u_{j+1}) = g^*(u_{j+1})$. Sinon, soit u_i le premier sommet de cette séquence à être pendant (en même temps que u_{j+1}). Le chemin $(u_o, u_1, \ldots, u_i)$ est optimal : $g(u_i) = g^*(u_i)$, donc :

$f(u_i) = g(u_i) + h(u_i) = g^*(u_i) + h(u_i)$, de plus h étant monotone :

$$\left.\begin{array}{l} h(u_i) \leqslant h(u_{i+1}) + K(u_i, u_{i+1}) \\ \vdots \\ h(u_l) \leqslant h(u_{j+1}) + K(u_l, u_{j+1}) \end{array}\right\} \quad h(u_i) \leqslant K(u_i, u_{i+1}) + \ldots + K(u_l, u_{j+1}) + h(u_{j+1})$$

$$f(u_i) \leqslant g^*(u_i) + K(u_i, u_{i+1}) + \ldots + K(u_l, u_{j+1}) + h(u_{j+1}) = g^*(u_{j+1}) + h(u_{j+1}) \quad f(u_{j+1})$$

Ce qui contredit le fait que $f(u_{j+1}) = \min \{ f(u) \mid u \in P \}$. □

Proposition 2.9. : Si h est monotone et coïncidente, alors A^* est de complexité en O(N).

Preuve : Immédiate à partir du résultat précédent : tout sommet est développé au plus une fois. □

Un autre corollaire intéressant permettra de déduire des propriétés pour la fonction f similaires à celle de h. On dit que :

f est monotone si $\forall u, v \in \Gamma(u)$: $f(u) \leqslant f(v)$;

f est minorante si $\forall u$: $f(u) \leqslant f^*(u)$;

f est coïncidente si $\forall w \in W$: $f(w) = f^*(w)$.

Proposition 2.10. : Si h est monotone et coïncidente, alors pour tout sommet u développé par A^*, f est monotone et coïncidente, et donc aussi minorante.

Preuve : Si u a été développé, $\forall v \in \Gamma(u)$: $f(v) = g(v)+h(v) \geq g^*(v)+h(v) = g^*(u) + K(u,v)+h(v)$. Or, $g^*(u) = g(u)$ et $K(u,v)+h(v) \geq h(u) \Rightarrow f(v) \geq g(u)+h(u) = f(u)$, donc f est monotone.

Par ailleurs, $\forall w \in W$, $f(w) = g(w)+h(w) = g^*(w)+0=f^*(w) \Rightarrow$ f est coïncidente. Finalement, $g(u) = g^*(u)$ et h minorante $\Rightarrow$ f est minorante. □

Remarquons que dans l'exemple 2.1., l'heuristique h est monotone (mais pas au sens de la définition restrictive de Pohl-Nilsson, du fait de $h(u_7) < h(u_3)$, $h(u_8) < h(u_7)$, ou $h(u_{10}) < h(u_{11})$). L'algorithme se comporte bien selon les propriétés des propositions 2.7. et 2.8., en particulier il développe à chaque itération le premier sommet de P.

Consacrons la fin de cette section à la discussion d'un autre aspect de la complexité d'A^*. On pourrait penser que cet algorithme, linéaire en N, a une complexité satisfaisante. En fait, il n'en est rien. Dans la plupart des applications, aussi bien du type optimisation, que du type résolution de problème, les données d'entrée d'A^* ne sont pas un graphe explicite G, mais un système de règles définissant implicitement G et caractérisant éventuellement les états terminaux. Le graphe G reflète alors l'espace de recherche du problème. Il s'agit, par exemple, d'un arbre de N = (n-1)!feuilles pour le problème du voyageur de commerce (sur n citées), d'un graphe de N = 9! = 362880 sommets pour le jeu très simple du "8-puzzle"[64], et de $N = 4,3.10^{19}$ pour un sous-ensemble d'états du cube hongrois[93]. La dimension N du graphe G (ou du sous-graphe fini G') est souvent telle que la caractérisation de la complexité d'A^* en fonction de N est peu réaliste, et ne reflète surtout pas un algorithme polynômial en la dimension de ses données d'entrée. D'où l'idée [224] d'adopter, comme paramètre caractéristique de la taille d'un problème de recherche, la longueur (en nombre d'arcs) du chemin optimal menant à un sommet terminal.

Soit M cette longueur. Un A^* disposant de l'information parfaite : $h = h^*$, explore uniquement les sommets sur le chemin optimal.

Ibaraki[99] montre que l'algorithme guidé par une heuristique presque-parfaite est aussi en O(M). Si h est seulement monotone, et pour un modèle simplifié d'une recherche dans un arbre m-aire uniforme, A^* est de complexité en $O(m^M)$, exponentielle en M.

Pour une heuristique minorante, mais proche de l'information parfaite, quelle serait la complexité de l'algorithme en fonction de l'écart (h^*-h) ?

Pohl, qui a initialisé la recherche dans cette direction, a montré [187,188] que A^* reste exponentiel en M, même si l'erreur relative $(h^*-h)/h^*$ est majorée par une constante. Par contre, si l'heuristique est au plus à une distance absolue de l'information parfaite, cet algorithme devient linéaire. Poursuivant dans ce sens, Gashnig (64) a établi les résultats suivants (r constante $\mathbb{R}^+$) :

- si $(1-r)h^* \leq h \leq h^*$, alors A^* est exponentiel en $O(m^{rM})$;
- si $h^* - \sqrt{h^*} \leq h \leq h^*$, alors A^* est exponentiel en $O(M\, m^m)$;
- si $h^* - \log h^* \leq h \leq h^*$, alors A^* est polynômial en $O(M^{\log m})$;
- si $h^* - r \leq h \leq h^*$, alors A^* est linéaire en $O(m^r M)$.

Une récente publication (Huyn, Dechter et Pearl[98]), en soulignant qu'une erreur absolue ou logarithmique était une qualité très rare pour un estimateur, que souvent une estimation se trouve entâchée d'au moins une erreur relative constante, et donc que les résultats précédents caractérisant la complexité en pire des cas d'A^* sont peu encourageants, s'est proposé d'analyser le comportement moyen de cet algorithme. Elle reprend le modèle de l'arbre uniforme m-aire, à coût unitaire, contenant une seule solution à la profondeur M, et suppose que les erreurs relatives $r = (h^*-h)/h^*$ en les divers noeuds de l'arbre sont des variables aléatoires indépendantes et uniformément réparties sur $[0, \beta \leq 1]$.

Deux résultats importants sont démontrés :

- si Prob $[r = 0] < (1 - 1/m)$, alors la complexité moyenne d'A^* est exponentielle ;

- pour tout ε donné $0 \leq \varepsilon \leq 1$ si Prob $[r > \varepsilon] > 1/m$, alors la complexité moyenne est encore exponentielle.

Pearl établit également (179) que la complexité moyenne d'A^* cesse d'être polynômiale dès que l'écart (h^*-h) croît plus rapidement que $\log h^*$. Un exemple pour illustrer ces résultats : l'exploration d'un arbre 4-aire (conforme au modèle de Pearl) n'évitera un comportement moyen exponentiel que si l'heuristique est exactement parfaite, $h = h^*$, en plus de 3 noeuds sur 4 !

Il semble donc que le comportement moyen de la recherche admissible ne soit pas d'un ordre de complexité inférieur à celui de son comportement en pire cas. Qu'en est-il de la recherche non admissible ?

Un autre résultat du à Pohl (188) montre que si on ne cherche plus une solution optimale, et qu'on essaie d'atteindre une solution quelconque, guidé uniquement par $f(u) = h(u)$, et si h est minorante avec écart à h^* borné : $(h^* - h) \leq r$, alors la complexité maximale de cette recherche est en $O(M.m^{2r})$; au lieu de $O(M.m^r)$ pour A^*. On aurait donc intérêt, pour réduire la complexité maximale, à rechercher une solution optimale, plutôt qu'une solution quelconque !

On défendra plutôt, dans ce qui suit, une approche intermédiaire entre les deux cas extrêmes de la recherche non admissible (qui ignore l'information $g(u)$ = coût du chemin $u_0 - u$) et de la recherche admissible.

°0°

2.3. RECHERCHE HEURISTIQUE ϵ- ADMISSIBLE DANS UN GRAPHE : A_ϵ

Un algorithme d'optimisation sera dit ϵ - admissible si, pour tout $\epsilon \geq 0$ donné comme paramètre, cet algorithme fournit une solution à écart relatif par rapport à une solution optimale majorée par ϵ . Deux arguments seront mis en avant pour justifier la nécessité d'une approche ϵ - admissible sur les problèmes de recherche heuristique dans un graphe.

Le premier argument reprend des résultats de (83,99) sur l'optimalité d'A^* : on établit en effet qu'une recherche guidée par une heuristique h minorante développe moins de sommets que tout autre algorithme admissible de recherche en profondeur ou en largeur, ou du type A^* qui utiliserait une heuristique h' moins informée que h (i.e., tel que $\forall u : h'(u) < h(u) \leq h^*(u)$).

Compte-tenu des résultats de la section précédente, il semble donc que l'approche admissible soit condamnée à avoir une complexité élevée, ou à rester confinée aux rares cas pour lesquels l'information heuristique est particulièrement pertinente.

Le deuxième argument remet en cause le critère qui est optimisé dans l'approche admissible et pour les applications du type résolution de problème. Initialement, A^* n'a pas été directement proposé pour une minimisation de chemin dans un graphe donné explicitement a priori (problème sur lequel on ne dispose pas, en général, d'information heuristique, et pour lequel bon nombre d'algorithmes satisfaisants sont connus, Cf. par exemple [77, Chap. 2]), mais plutôt pour guider l'interpréteur d'un système de règles entre un état initial et un état terminal décrit comme objectif. L'efficacité de l'interpréteur pour un chemin menant à un état terminal est aussi importante que le coût de ce chemin, seul pris en compte par A^*. Dans l'exemple de la génération de plan en robotique, le plan généré est destiné à être appliqué par le robot un nombre très limité de fois, et souvent une seule fois. De plus, le coût d'exécution d'un plan est du même ordre de grandeur que son coût de génération. Il est donc manifeste que l'efficacité (algorithmique) du générateur de plan est au moins aussi importante que l'efficacité du plan généré.

Un autre exemple fréquent dans la littérature, est celui de la résolution de puzzles et de jeux : même si la règle du jeu consiste à atteindre l'état but le plus rapidement possible, le temps "réflexion", s'il est compté, sera souvent primordial.

Il ne s'agit cependant pas de tomber dans l'excès inverse en affirmant que le seul critère digne d'attention est le temps de recherche d'une solution.

On défendra plutôt l'approche d'une pondération, contrôlée par l'utilisateur, entre les deux critères : temps de recherche d'une solution - coût de la solution trouvée.

La dualité des deux objectifs précédents n'a pas été très clairement prise en compte dans la littérature. Initialement, l'heuristique h a été introduite autant pour accélérer la recherche, que pour minimiser le critère de coût. Nilsson(170)laisse entendre confusément que complexité d'A^* et coût $K^*(u,w)$ sont deux objectifs concomittants qu'on optimise simultanément. L'utilisation, dans cette référence, d'une distribution de coût unitaire ($k(u,v) = 1$), ne contribue pas à la clarification (plus une solution est coûteuse, plus l'algorithme développera de sommets sur le chemin qui y mène). Ceci, d'autant plus que la complexité et l'optimalité d'A^* sont analysées(170) uniquement par rapport au nombre de sommets distincts développés, sans tenir compte du nombre des développements de chacun ! Le résultat sur la recherche non admissible, précédemment citée, est également révélateur de cette confusion coût-complexité (la recherche d'une solution optimale minimiserait la complexité).

De mon point de vue, l'optimalité du coût et la complexité de la recherche sont le plus souvent contradictoires, et la relaxation de l'objectif d'optimalité en un objectif d'ε- optimalité, permet de réduire la complexité.

Le principe de base de la recherche ε- admissible est de disposer à chaque itération d'un large choix de sommets pouvant être développés (au lieu d'être contraint, comme A^*, de poursuivre uniquement sur le premier sommet pendant $\hat{u}$), et de prendre parmi eux celui qui réduira la complexité de l'algorithme.

Par définition, on dira qu'un sommet pendant u est acceptable, relativement à l'objectif d'optimalité, s'il vérifie : $f(u) \leq (1+\varepsilon)\ f^*(u_o)$. Un algorithme ϵ-admissible pourra développer tout sommet pendant acceptable · Il devra choisir celui qui minimise la complexité de la recherche.

Pour guider ce choix, on définit une deuxième fonction heuristique $h_c(u)$: estimation du nombre minimal $h_c^*(u)$ d'itérations nécessaires pour atteindre un sommet terminal à partir de u. C'est donc un indicateur de la longueur, en nombre d'arcs, du chemin le plus court entre u et un sommet terminal. Cet estimateur est, en général, indépendant de la distribution de coût K.

Analysons informellement diverses stratégies possibles de choix du sommet pendant acceptable qui sera développé :

- Stratégie opportuniste : développer à chaque itération le sommet pendant acceptable à h_c minimal.

Dans le cas idéal d'une heuristique parfaite $h_c = h_c^*$, tout sommet possède au moins un successeur v tel que $h_c(v) = h_c(u) - 1$. Si u a été développé à l'itération j, à l'itération suivante, on développera ce successeur v de u, tant que v reste acceptable. Donc, l'algorithme poursuivra le développement d'un même chemin dans le graphe et n'abandonnera ce chemin que pour une des deux raisons suivantes :

1) Soit que les successeurs acceptables v de u sont tel que $h_c(v) > h_c(u)$; ce qui sera le cas si h_c n'est pas localement consistente ;

2) Soit que u n'a pas de successeur acceptable.

S'il n'y a pas de corrélation directe entre coût et longueur d'un chemin, la première situation pourraît être fréquente. Dans ce cas, le choix systématique du sommet acceptable à h_c minimal, qui conduirait l'algorithme à changer de chemin, l'obligerait, quelques itérations plus loin, à abandonner ce deuxième chemin dès qu'un autre apparaîtrait, promettant d'être un tant soit peu plus court. La tendance à favoriser le "backtracking" sera d'autant plus prononcée que l'heuristique h_c sera imparfaite.

La stratégie opportuniste risque donc de conduire à l'exploration de nombreux chemins différents et à de nombreuses itérations sur un même chemin, et donc, par là, à être pénalisante sur le plan de la complexité.

- Stratégie de développement en profondeur : poursuivre le développement d'un même chemin tant que le coût de ce chemin reste acceptable. Parmi les fils acceptables du sommet qui vient d'être développé, on pourrait choisir de poursuivre soit sur celui à f minimal, soit sur celui à h_c minimal, soit sur un fils offrant un compromis entre f et h_c.

Remarquons que la stratégie de développement en profondeur rejoint la stratégie opportuniste dans le cas d'une corrélation directe coût-longueur d'un chemin : si u a été développé et possède un fils acceptable, celui-ci sera nécessairement le sommet à h_c minimal. Un exemple de cette situation (en dehors du cas trivial de la distribution de coût unitaire K(u,v) = 1) est celui de l'exploration d'un arbre (ou d'un latticiel dédékindien) de hauteur maximale finie: on pourrait prendre $\forall v \in \Gamma(u)$: $h_c(v) = h_c(u) - 1$; et $h_c(u_o)$ = hauteur maximale de l'arbre.

La stratégie de développement en profondeur permet de s'affranchir du premier cas de backtracking de la stratégie opportuniste. Peut-on s'affranchir également du deuxième cas (absence de successeur acceptable) ? Dans de nombreuses situations, la réponse est affirmative. En effet, comme on ne connaît pas $f^*(u.)$, l'acceptabilité d'un sommet est jugée par rapport à f(û), minorant de $f^*(u.)$. Il suffira, éventuellement, de développer û pour élever le seuil d'acceptabilité. D'où :

- Stratégie persévérante : Si v, le meilleur fils du sommet u qui vient d'être développé, n'est pas acceptable ; mais que l'écart au seuil d'acceptabilité est faible, alors développer û, puis revenir au chemin précédemment exploré si v est devenu acceptable relativement au nouveau seuil.

On peut, bien entendu, procéder au développement d'une séquence de plusieurs premiers sommets pendants $\hat{u}$, en vue de rendre un chemin de nouveau acceptable. Cette acceptabilité est d'autant plus plausible dans le cas d'une heuristique h monotone, car alors la séquence $f(\hat{u})$ est non décroissante (proposition 2.7.)

Remarquons qu'ici, le développement du sommet $\hat{u}$ diffère des autres développements sur le plan de l'objectif recherché, qui n'est plus d'atteindre un sommet terminal, mais d'élever le seuil d'acceptabilité. C'est un investissement pour réduire la complexité de l'algorithme, mais qui est également utile à la progression de la recherche : le développement de $\hat{u}$ ne sera pas perdu si le chemin en cours d'exploration n'aboutit pas et qu'un "backtracking" doit intervenir.

L'ensemble de ces considérations conduit à l'algorithme A_ϵ dans lequel, en plus des notations P, Q et $\hat{u}$ précédemment définies, on notera :

- $\hat{u}_\epsilon$: le premier fils (au sens de l'ordre précédent) du sommet u qui vient d'être développé ;
- λ : meilleur chemin <u>complet</u> atteint par l'algorithme ;
- Y : coût du chemin λ ;
- $y = \max \{f(\hat{u})\}$: plus grand des minorants de $f^*(u_0)$.

La deuxième stratégie ramène à la première bifurcation acceptable, avec le risque d'entraîner un développement en largeur à ce niveau (nombreux développements coûts et "backtrackings").

La troisième stratégie exige une expérimentation préalable pour déterminer les facteurs de pondération. On retrouve, ici, le fameux dilemne du "backtracking", qu'il semble difficile de résoudre à priori et d'une façon générale. En l'absence d'une bonne connaissance sur les fiabilités relatives de f et de h_c et d'informations heuristiques supplémentaires sur l'application considérée qui permettraient de trancher, on pourraît se baser sur le fait que la première stratégie est algorithmiquement la moins complexe à mettre en oeuvre, et surtout ne nécessite pas la connaissence de h_c.

2) CONDITION : est une procédure retournant une valeur logique $\{$ Vrai, Faux $\}$, et permettant de déterminer l'opportunité d'application de la stratégie persévérante. On peut se baser sur les considérations suivantes :

- le nombre maximal d'itérations successives de (2.5.) doit rester très faible (2 ou 3) ;

- le chemin en cours de développement est presque acceptable, i.e. : $f(\hat{u}_\varepsilon) - (1+\varepsilon)y \leq$ seuil ;

- ce chemin est suffisamment proche d'une solution pour mériter qu'on y persévère, i.e., $h_c(u)$ est faible ;

- éventuellement, il existe un chemin complet λ, presque acceptable : $y - (1+\varepsilon)y \leq$ seuil ;

- $\hat{u}$ correspond au seuil, i.e. : $f(\hat{u}) = y$;

- h est monotone ;

- le premier sommet pendant $\hat{u}$ est assez isolé en tête de P, i.e., Y ou $f(\hat{u}_\varepsilon)$ seraient acceptables relativement aux 2ème ou 3ème sommets pendants.

D'autres facteurs, tels que $\hat{u}$ très éloigné d'une solution ($h_c(\hat{u})$grand) pourront encourager dans le sens du développement de $\hat{u}$.

Ici, il est nécessaire de montrer que la stratégie persévérante n'est pas équivalente à un "backtracking" systématique dès que le chemin exploré cesse d'être acceptable. En effet :

- si le "backtracking" a lieu sur $\hat{u}$, l'algorithme ne reprendra le développement du chemin laissé en suspens que lorsque ce chemin redeviendra le meilleur pendant, alors qu'il aurait pû être acceptable et déboucher sur une solution bien avant ;

- si le "backtracking" a lieu sur le sommet pendant acceptable à h_c minimal, A_ε pourraît avoir à effectuer de courtes explorations sur de nombreux chemins, bloqué par un $\hat{u}$ ayant un $h_c(\hat{u})$ élevé, et ceci jusqu'à ce qu'il n'y ait plus de sommet acceptable autre que $\hat{u}$ lui-même.

Par ailleurs, le backtracking systématique ne tient pas compte de l'existence de solution complète connue par l'algorithme, alors que quelques itérations sur (2.6.1.) peuvent rendre un chemin λ acceptable et conduire à l'arrêt d'A_ε.

Donc, cette stratégie offre une plus grande souplesse. Remarquons que prendre

CONDITION = Faux pour toute itération (i.e., supprimer la boucle 2.5.), revient à un backtracking systématique sur le sommet SELECTION (P). Finalement, si SELECTION (P) = $\hat{u}$, l'application de l'itération (2.5.) développera le sommet $\hat{u}$ sur lequel l'algorithme aurait,de toute manière, "backtracké ": la stratégie ne conduit pas à des développements inutiles.

L'initialisation de l'algorithme développe la racine u_0. L'itération (2) est la boucle principale. Après le développement d'un sommet u (en 2.3) et si $\hat{u}_\epsilon$, meilleur fils de u n'est pas acceptable, on teste l'opportunité d'appliquer la stratégie persévérante (itération 2.5). Si $\hat{u}_\epsilon$ est acceptable ou s'il l'est devenu après (2.5.), l'itération suivante le développera en poursuivant donc sur le même chemin. Sinon, on procèdera (en 2.2.) à un backtracking en repartant d'un autre chemin.

En dehors d'un test de terminaison, le développement d'un sommet est identique à celui effectué par A^*. Si un fils de u est terminal et si le chemin correspondant est le meilleur atteint jusqu'à ce point, on retient ce chemin (λ) et son coût (y).

Aussi bien la boucle principale d'A_ϵ que la boucle interne (2.5.) s'arrêtant dès qu'il n'y a plus de sommet pendant, ou que le coût du meilleur chemin complet connu est devenu acceptable. Cette dernière condition peut se produire soit en poursuivant un chemin acceptable jusqu'à la terminaison, soit en élevant le seuil d'acceptabilité.

En plus de DEVELOPPEMENT, A_ϵ fait appel à deux procédures supplémentaires :

1) SELECTION : Si un "bactracking" doit intervenir (i.e., $\hat{u}_\epsilon$ n'est pas acceptable), cette procédure choisit dans P un sommet particulier sur lequel A_ϵ poursuivra (et commencera à la première itération).

Plusieurs stratégies sont possibles :

i) SELECTION (P) ← $\hat{u}$, i.e., somment pendant à f(u) minimal ;

ii) SELECTION (P) ← un sommet pendant acceptable à h_c minimal ;

iii) SELECTION (P) ← un sommet pendant acceptable minimisant ($\alpha f(u) + \beta h_c(u)$).

La première stratégie fait remonter l'algorithme le plus "haut" possible avec l'espoir de repartir sur un bon chemin (qui resterait longtemps acceptable, car on espère élever le seuil $(1 + \epsilon)$ y).

Algorithme A_ϵ

Données d'entrée : $G = (U, \Gamma)$, u_o, W, coûts $K \geqslant 0$, fonctions h et h_ϵ, paramètre $\epsilon \geqslant 0$.

1. Initialisation : $P \leftarrow \{u_o\}$; $Q \leftarrow \emptyset$; $Y \leftarrow \infty$
   $g(u_o) \leftarrow 0$; $y \leftarrow f(u_o) \leftarrow h(u_o)$
   DEVELOPPEMENT (u_o)
   $\hat{u}_\epsilon \leftarrow \hat{u}$

2. Itérer tant que $[P \neq \emptyset$ et $Y > (1+\epsilon)y]$

2.1. Si $f(\hat{u}_\epsilon) \leqslant (1+\epsilon)y$ alors $u \leftarrow \hat{u}_\epsilon$

2.2. Sinon $u \leftarrow$ SELECTION (P)

2.3. DEVELOPPEMENT (u)

2.4. $\hat{u}_\epsilon \leftarrow$ 1er fils pendant de u, dans l'ordre f croissant, g décroissant

2.5. Itérer tant que $[P \neq \emptyset$ et $\min\{Y ; f(\hat{u})\} > (1+\epsilon)y$ et CONDITION $]$
   DEVELOPPEMENT $(\hat{u})$
   Fin itération 2

Fin itération 2

3. Fin.

Données de sortie : Si $Y < \infty$, alors sortir le chemin λ et $\epsilon \leftarrow (Y-y)/y$
Sinon $(P = \emptyset)$: il n'y a pas de solution.

Pour développer un sommet u, l'algorithme fait appel à la procédure :

DEVELOPPEMENT (u)

1. Supprimer u de P et le mettre dans Q

2. Itérer sur $\{ v \in \Gamma(u) \mid (v \notin P$ et $v \notin Q)$ ou $g(v) > g(u) + K(u,v)\}$

2.1. $g(v) \leftarrow g(u) + K(u,v)$

2.2. $f(v) \leftarrow g(v) + h(v)$

2.3. $\gamma(v) \leftarrow u$

2.4. Ranger v dans P suivant l'ordre f croissant, g décroissant

2.5. Si $(v \in W$ et $f(u) < Y)$, alors faire :
   $Y \leftarrow f(v)$
   $\lambda \leftarrow$ chemin $(u_o\text{-}v)$ obtenu en suivant les pointeurs γ^{-1}.

2.6. Fin itération 2.

3. $y \leftarrow \max\{y, f(\hat{u})\}$

4. Fin

Il est intéressant de comparer les conditions d'arrêt de A_ε avec celle de A^*. L'algorithme A^* s'arrête lorsque, au moment de développer le meilleur sommet pendant, il s'aperçoit que ce sommet est terminal. A_ε par contre, surveille les sommets terminaux au moment de leur génération (ou réorientation de leur pointeur γ^{-1}).

En conservant la meilleure solution atteinte à une certaine étape, cet algorithme peut, non seulement, s'arrêter dès que l'estimation de l'optimum permet de garantir l'acceptabilité de cette solution, mais, de plus, il peut exploiter l'information heuristique (h et h_c) disponible pour orienter sa recherche vers un tel arrêt. En imageant la comparaison, on peut dire qu'A^* s'arrête en butant par surprise sur une solution, alors qu'A_ε s'y dirige grâce à une vision plus profonde de l'espace exploré. Un autre avantage d'A_ε est de pouvoir fournir, si nécessaire, plusieurs solutions pour le prix d'une seule (et dont plus d'une peut être acceptable) : il suffit, pour cela, de transformer λ en une structure de pile.

Par ailleurs, on peut également demander à l'algorithme de fournir une solution sans backtracking (simple recherche en profondeur ne développant que les sommets sur le chemin solution) : il suffit, pour cela, de fixer $\varepsilon = \infty$; A_ε retournera un $\varepsilon' \ll \varepsilon$, ε' pouvant être amélioré ultérieurement par la stratégie persévérante si cela est nécessaire.

Remarque : On peut rapprocher la stratégie persévérante de la technique d'exploration d'un arbre de jeux utilisée dans l'algorithme B^* (14) qui explore certains sommets dans le seul but de confirmer la validité de la solution (partielle ou complète).

Illustrons le comportement de l'algorithme sur un exemple :

Exemple 2.2. : Appliquons A_ε avec SELECTION (P) = Ω et CONDITION = Faux sur le graphe G de l'exemple 2.1., avec $h_c(u) = 0, \forall u$.

Pour $\varepsilon = 0{,}5$, la séquence d'itérations d'A_ε est la suivante :

Sommet u développé	Successeurs générés	Ensemble P	$(1+\varepsilon)y$	y	$\hat{u}_\varepsilon$
u_o (19)	u_1, u_2, u_3, u_4	$u_3(5+14)$; $u_1(3+16)$; $u_2(4+16)$; $u_4(6+15)$	28.5	∞	$u_3(19)$
u_3 (19)	u_6	u_1 ; u_2 ; u_4 ; $u_6(14+9)$	28.5	∞	$u_6(23)$
u_6 (23)	u_7, u_{10}, u_{11}	u_1 ; u_2 ; u_4 ; $u_{10}(20+3)$; $u_7(16+7)$; $u_{11}(16+8)$	28.5	∞	$u_{10}(23)$
$u_{10}(23)$	$u_{12} \in W$	u_1 ; u_2 ; u_4 ; u_7 ; u_{11}	28.5	24	$u_{12}(24)$
Sortie de $\varepsilon' = (24-19)/19 = 0.26$ et du chemin $\lambda = (u_o, u_3, u_6, u_{10}, u_{12})$					

Pour $\varepsilon = 0.25$, A_ε reprend les trois premières itérations du tableau précédent et poursuit sur :

Sommet u développé	Successeurs générés	Ensemble P	$(1+\varepsilon)y$	y	$\hat{u}_\varepsilon$
$u_{10}(23)$	$u_{12} \in W$	u_1; u_2; u_4 ; u_7 ; u_{11}	23.75	24	$u_{12}(24)$
u_1 (19)	$\emptyset$	u_2; u_4; u_7 ; u_{12} ; u_{11}	25	24	u_2 (20)
Sortie de $\varepsilon' = (24-20)/20 = 0.2$ et du chemin $\lambda = (u_o, u_3, u_6, u_{10}, u_{12})$					

Alors qu'A^* nécessite 9 itérations sur cet exemple, A_ε fournit une solution majorée à 26 et 20 % de l'optimum en 4 et 5 itérations respectivement. Remarquons que l'écart réel de cette solution à l'optimum est de 4.3 % seulement.

Etablissons, à présent, quelques propriétés de l'algorithme A_ε. Pour démontrer sa convergence (i.e., que sur un graphe fini l'algorithme s'arrête soit avec une solution, soit avec $P = \emptyset$ s'il n'y a pas de solution), il suffit de reprendre la preuve de la proposition 2.1. qui demeure valide. L'ε-admissibilité est donnée par la proposition suivante :

Proposition 2.11. : Si G est un δ- graphe fini ou infini à coûts positifs ou nuls et dans lequel il existe au moins un chemin solution de longueur finie, et si h est une heuristique minorante, alors $\forall \varepsilon \geq 0$, A_ε est ε'- admissible, i.e., il fournit une solution à écart relatif par rapport à l'optimum majoré par ε', $\varepsilon' \leq \varepsilon$.

Preuve : On vérifie facilement que la proposition 2.2. reste valide ici : à toute itération d'A_ε, il existe un sommet pendant u_i tel que : $f(u_i) \leq f^*(u_o)$. De plus, à chaque itération, le sommet u développé est acceptable : $f(u) \leq (1+\varepsilon)y$; or $y = \max\{f(\hat{u})\} \leq f(u_i) \leq f^*(u_o) \Rightarrow f(u) \leq (1+\varepsilon)\, f^*(u_o)$.

Par ailleurs, si G_ε est le sous-graphe de G engendré par l'ensemble des sommets $\{u \in U \mid g^*(u) \leq (1+\varepsilon)\, f^*(u_o)\}$, d'une part G_ε est nécessairement fini ; d'autre part, tout sommet $u \notin G_\varepsilon$ ne peut être développé par A_ε d'après ce qui précède, car pour un tel sommet $f(u) > (1+\varepsilon)\, f^*(u_o)$. Le sous-graphe G_ε contient nécessairement au moins un sommet terminal w. A_ε explore éventuellement tous les chemins menant à tous les sommets non terminaux de G_ε, et s'arrête après avoir accédé à W et établi son acceptabilité. Par définition de ϵ' :

$$Y = f(w) = (1+\varepsilon')y \leq (1+\varepsilon)y \leq (1+\varepsilon)\, f^*(u_o).$$

□

Remarques :

1) Contrairement à certains schémas d'approximation (par exemple (102)), l'ε- admissibilité d'A_ε est établie même pour $\varepsilon = 0$. Dans ce cas limite, l'algorithme est donc admissible et fournit une solution exactement optimale. De plus, moyennant certaines contraintes très simples sur les procédures SELECTION et CONDITION, A_ε procèdera pour $\varepsilon = 0$ au même nombre d'itérations et développera les mêmes sommets (sauf le dernier) et dans le même ordre qu'A^*.

2) Dans le pire cas, les deux algorithmes A_ε et A^* explorent respectivement tous les sommets des sous-graphes G_ε et G' (engendré par $\{u \in U \mid g^*(u) < f^*(u_0)\}$). Or, d'une part G' est strictement inclus dans G_ε et, d'autre part, le nombre de développement d'un même sommet n'est pas restreint davantage dans A_ε que dans A^*. La complexité en pire cas d'A_ε, n'est donc malheureusement pas inférieure à celle d'A^*.

Heuristique monotone : La monotonie de l'heuristique entraîne peu de propriétés intéressantes pour A_ε. En effet, les propositions 2.7. et 2.8. ne sont plus valides. Elles se traduisent ici par les faits que si h est monotone, la séquence $f(\hat{u})$ est non décroissante (à prendre en compte dans CONDITION) et $g(\hat{u}) = g^*(\hat{u})$ à tout moment dans l'algorithme. On pourra donc remplacer y par la valeur courante de $f(\hat{u})$. De plus :

Propositions 2.12. : Si h est monotone, alors pour tout sommet u développé par A^*, on a : $g^*(u) \leq g(u) \leq g^*(u) + \varepsilon f^*(u)$.

Preuve : Soit $(u_0, u_1, \ldots, u_i, u)$ un chemin optimal menant à u, et u_i le premier sommet de ce chemin à se trouver pendant au moment du développement de u.

$f(u) \leq (1+\varepsilon)\ f(\hat{u})$; $f(\hat{u}) \leq f(u_i)$; et $g(u_i) = g^*(u_i)$; d'où

$g(u) + h(u) \leq (1+\varepsilon)\ (g^*(u_i) + h(u_i))$

$h(u_i) - h(u) \leq K(u_i, u_{i+1}) + \ldots + K(u_1, u)$; d'où :

$g(u) \leq (1+\varepsilon)\ (g^*(u_i) + h(u_i)) - h(u)$

$g(u) \leq (1+\varepsilon)\ [g^*(u_i) + K(u_i, u_{i+1}) + \ldots + K(u_1, u) + h(u)] - h(u)$

$g(u) \leq (1+\varepsilon)\ g^*(u) + \varepsilon h(u) \leq g^*(u) + \varepsilon f^*(u)$

□

On peut déduire de ce résultat que, contrairement à A^* qui, en cas de monotonie, ne développe un même sommet u qu'au plus une fois, A_ε pourra développer u autant de fois qu'il existe de chemins menant à u, et ayant des coûts distincts et compris entre $[g^*(u),\ g^*(u) + \varepsilon f^*(u)]$. A titre simplement indicatif, la complexité en pire cas d'A_ε sur des graphes valués par des entiers, est en O(bN) avec $b = 1 + g^*(u) + \varepsilon f^*(u) - g^*(u) = 1 + \varepsilon f^*(u)$; or $f^*(u) \leq f(u) \leq (1+\varepsilon)\ f^*(u_0) \leq (1+\varepsilon)\ Y$; d'où : $b \simeq 1 + \varepsilon\ (1+\varepsilon)\ Y$. On retrouve pour $\varepsilon = 0$, la complexité en O (N) d'A^*.

Heuristique α- minorante : Pour de nombreuses applications, l'information disponible, à priori, ne permet que très difficilement la construction d'une fonction d'évaluation heuristique ayant les bonnes propriétés requises de monotonie ou même de minoration. Dans certains cas (dont on verra un exemple en section 3.3.3.), ces propriétés ne sont accessibles qu'au prix d'une complexité de calcul de l'heuristique très élevée.

De plus, plusieurs auteurs ont observé qu'une heuristique minorante est souvent moins performante, en nombre de sommets développés, qu'une heuristique non minorante (64) . En effet, pour garantir $h(u) \leq h^*(u)$ pour tout $u \in U$, on est conduit à prendre h assez loin de h^*, alors qu'une estimation plus proche de h^* est possible si on relaxe la contrainte, en acceptant par exemple que h(u) ne dépasse pas $h^*(u)$ de plus d'une certaine valeur.

Harris (81) a exploré cette approche ("Bandwidth Search"), en montrant que A^* avec h tel que $\forall u \in U$: $h(u) \leq h^*(u) + e$, fournit une solution à écart absolu de l'optimum borné par e.

Moyennant une concession supplémentaire sur le degré d'optimalité, on peut profiter à la fois d'une heuristique plus pertinente et de stratégies de recherche plus directives. En effet, le résultat précédent se généralise très facilement à A_ε . On dira que :

h est α-minorante si $\forall u \in U$: $h(u) \leq (1+\alpha)\, h^*(u)$; et on a :

Proposition 2.13. : Pour tout $\varepsilon \geq 0$; l'algorithme A_ε avec une heuristique α- minorante est ε'- admissible, et $\varepsilon' \leq \varepsilon + \alpha\,(1+\varepsilon)$.

Preuve : On reprend celle de la proposition (2.11), en notant que :

$f(u_i) = g^*(u_i) + h(u_i) \leq g^*(u_i) + (1+\alpha)\, h^*(u_i)$; d'où : $y \leq (1+\alpha)\, f^*(u_o)$;

et $Y \leq (1+\varepsilon)\,(1+\alpha)\, f^*(u_o)$. □

Si on désire une solution garantie à mieux que ε_o de l'optimum, il suffit d'appeler A_ε avec $\varepsilon = (\varepsilon_o - \alpha)/(1+\alpha)$. Notons qu'on ne peut avoir une garantie meilleure que α.

On vérifie, par ailleurs, que si h est à écart constant de h^* : $h(u) \leq h^*(u) + e$, alors la solution fournie par A_ε est telle que $f(w) \leq (1+\varepsilon)\ (e+f^*(u_o))$.

Complexité d'A_ε : Donnons un dernier résultat sur la complexité en pire cas d'A_ε. Dans le modèle de l'arbre uniforme m-aire à coûts unitaires et avec une heuristique minorante, le facteur b de la complexité en O(bN) n'intervient plus : l'absence de cycle dans un arbre garantit le développement d'un même sommet au plus une fois. Néanmoins, dans le cas d'une erreur relative bornée : $(1-r)h^* \leq h \leq h^*$, et si SELECTION (P) = $\hat{u}$ et CONDITION = Faux, on peut établir qu'en pire cas A_ε est de complexité en $O\ (m^{rM}(1+\varepsilon M))$, (alors qu'$A^*$ était en $O\ (m^{rM})$). Indiquons les grandes lignes de cette analyse (assez longue) sans entrer dans les détails :

- l'hypothèse simplificatrice d'un sommet terminal unique dans l'arbre (faite dans le cas d'A^*) n'est plus acceptable pour une recherche ε- admissible. Aussi, on suppose qu'il existe une solution unique à la profondeur M (l'optimum); (m-1) solutions à la profondeur (M+1) ; m (m-1) à la profondeur (M+2) ; ..., etc, ces sommets terminaux étant répartis de telle sorte que tout sommet dans l'arbre se trouve au plus à une distance M d'une solution (l'arbre étant infini).

- étant donné les coûts unitaires et pour garantir l'existence de plus d'une solution ε- admissible, on suppose $\varepsilon > 1/M$.

- on suppose que les valeurs de l'heuristique h sont telles que l'algorithme développe un nombre maximal de sommets (stratégie de l'adversité). Ceci conduit à prendre pour tout sommet u à une distance M d'une solution $h(u) = (1-r)\ h^*(u) = (1-r)M$. On vérifie que pour un sommet u situé sur un bon chemin ($h^*(u) < M$), $h(u) = (1-r)\ M$ est plus désavantageux que $h(u) = h^*(u)$. Enfin, face à des alternatives identiques (même f), l'algorithme prend toujours le pire choix.

La complexité d'A_ε est donc en pire cas supérieure à celle d'A^*, même pour ce modèle dans lequel un même sommet est développé au plus une fois. Ceci est dû au fait que le sous-graphe (sous-arbre ici) fini G_ε exploré par l'algorithme A_ε contient strictement celui G' auquel est restreint A^* : sur une fausse voie A_ε descend plus profondément qu'A^*. L'analyse en pire cas, qui oriente systématiquement vers une fausse voie, est donc très défavorable à la recherche ε-admissible, seule une analyse de la complexité moyenne permettra d'établir l'intérêt d'A_ε relativement à A^*.

Aussi bien l'approche empirique de Gaschnig (64) que le modèle formel de Huyn-Pearl(98), ont montré qu'A^* avait un comportement moyen de même ordre de complexité que son comportement en pire cas. Un modèle empirique établi sur un algorithme similaire à A_ε (chapitre 4) suggère, pour la recherche ϵ - admissible, un comportement moyen beaucoup plus favorable que le pire cas. Une analyse formelle reste à faire ; elle présente, néanmoins, de nombreuses difficultés, parmi lesquelles l'absence d'une condition nécessaire et suffisante de développement d'un sommet (celle très simple d'A^* : $f(u) \leq f^*(u_o)$ n'étant pas vraie pour A_ε).

°0°

2.4. RECHERCHE HEURISTIQUE DANS UN HYPERGRAPHE : AO*

Dans le modèle considéré tout au long des deux sections précédentes, chaque règle de décision est un opérateur de changement d'état. Une solution à un problème qui se pose dans un état u_o est une séquence linéaire d'opérateurs-règles de décision (i.e., un chemin dans le graphe G) qui amène le système de u_o à un état terminal.

Un modèle plus riche donne à chaque règle, outre le rôle d'opérateur de changement d'état, celui d'opérateur de décomposition. Une règle R transforme un état u en un ensemble d'états $\{u_1, \ldots, u_k\}$: le problème qui se pose dans l'état u est ramené, après l'application de R, à la résolution de chacun des problèmes correspondant aux états $u_1, \ldots, u_k$. La solution du problème initial u_o n'est plus purement séquentielle, mais s'enrichit d'une structure parallèle : elle correspond à l'application d'une certaine règle à u_o, puis à l'application simultanée, ou dans un certain ordre, des règles correspondantes aux états successeurs de u_o, ... etc, jusqu'à ce que tous les états atteints soient terminaux.

Un modèle équivalent, appelé dans (171) "Système de règles décomposables", dissocie l'opération de décomposition d'un état (décomposition considérée comme propre à cet état), de l'opération de transformation d'état résultant de l'application d'une règle.

La structure de recherche correspondante à ces deux modèles est usuellement décrite par un arbre ET/OU. Dans l'état u_o partent autant d'arêtes qu'il y a de règles applicables à u_o (noeud OU de l'arbre). Un noeud successeur de u_o correspond à une règle particulière R dont l'application transforme et décompose u_o en l'ensemble $\{u_1, \ldots, u_k\}$ (noeud ET).

Pour tenir compte des états identiques provenant de l'application de règles distinctes, et pour éviter de les dupliquer dans la représentation, on passe du modèle de l'arbre ET/OU à un modèle par graphe ET/OU, ou plus avantageusement à une représentation par un hyperlatticiel.

Quelques extensions aux définitions usuelles sur les hypergraphes sont nécessaires :

- $H = (U, \mathcal{E})$ est un hypergraphe orienté, ssi chaque hyperarête $(v_o, v_1, \ldots, v_k)$ est un sous-ensemble de U partiellement ordonné en un sommet père v_o et k sommets fils $v_1, \ldots v_k$. Une telle hyperarête sera dite connecteur issu de v_o.

L'ensemble des connecteurs issus d'un sommet u est indicé sur $\{1 \leq j \leq J(u)\}$, et on notera par $\Gamma_j(u)$ à la fois le $j^{ème}$ connecteur et l'ensemble des fils de u le long de ce connecteur. De plus, on désignera par :

$$\Gamma_H(u) = \bigcup_{1 \leq j \leq J(u)} \Gamma_j(u) \; ; \; \Gamma_H^{-1}(u) = \{ v \in U \mid \exists\, v, \Gamma_j(v) \text{ avec } u \in \Gamma_j(v) \}$$

$\hat{\Gamma}_H(u)$ et $\hat{\Gamma}_H^{-1}(u)$ sont les fermetures transitives des relations précédentes.

. H possède une racine u_o, ssi $\{u_o\} \cup \hat{\Gamma}_H(u_o) = U$

. H est sans circuit ssi $\forall u : u \notin \hat{\Gamma}_H(u)$

. un hyperlatticiel est un hypergraphe orienté, sans circuit, et muni d'une racine.

Remarque : On peut associer à un hyperlatticiel H le graphe $G_H = (U, \Gamma_H)$ obtenu en remplaçant chaque connecteur $\Gamma_j(u)$ de H par l'ensemble d'arcs $\{(u,v) \mid v \in \Gamma_j(u)\}$ et en confondant les arcs multiples. On vérifie alors que G_H est un latticiel.

Dans un problème de recherche heuristique représenté par un hyperlatticiel $H = (U, \mathcal{E})$, la racine u_o est l'état initial du système. Chaque connecteur issu d'un sommet u représente une règle applicable à u. Le sous-ensemble $W \subset U$ des sommets terminaux correspond aux états réalisant l'objectif à atteindre.

On suppose, à ce niveau, que tout sommet u est soit terminal, soit possède au moins un connecteur qui en est issu.

Une solution à un problème de recherche se posant dans l'état u_o est un sous-latticiel (SL) de H, noté Λ et défini par : (i) u_o est un sommet de Λ ;

et (ii) pour tout sommet v de Λ., ou bien v est terminal, ou bien il existe dans H un connecteur unique $\Gamma_j(v)$ tel que : 1) pour tout $w \in \Gamma_j(v)$, w est fils de v dans Λ et, 2) v n'admet pas dans Λ d'autre fils que les sommets de $\Gamma_j(v)$.

Exemple 2.3. : L'hyperlatticiel de la figure 2.2. suivante contient 24 sommets et 26 connecteurs (représentés par des groupes d'arêtes reliées par un double trait). Ainsi, les connecteurs issus de u_o sont :

$\Gamma_1(u_o) = \{u_1, u_2\}$; $\Gamma_2(u_o) = \{u_8\}$ et $\Gamma_3(u_o) = \{u_3, u_4\}$. Les sommets terminaux sont $W = \{u_4, u_{15}, u_{16}, u_{17}, u_{20}, u_{21}, u_{22}, u_{23}\}$.

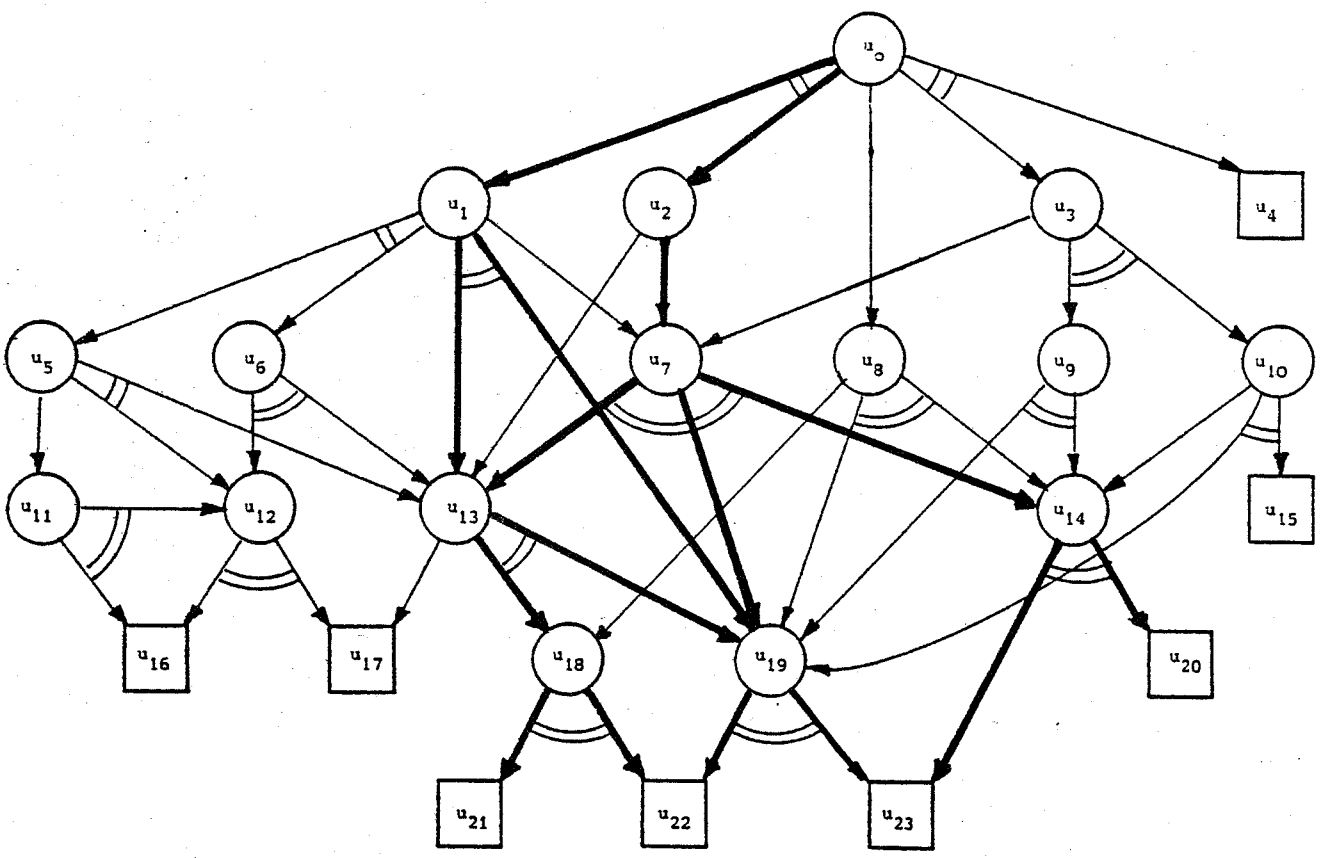


FIGURE 2.2.

Un exemple (marqué en trait renforcé) d'un sous-latticiel issu de u_o est $\Lambda = \{u_o, u_1, u_2, u_7, u_{13}, u_{14}, u_{18}, u_{19}, u_{20}, u_{21}, u_{22}, u_{23}\}$.

Une procédure (non déterministe) permettant de définir un SL de H à partir de sa racine u_o pourrait être la suivante (le symbole U désigne une union entre graphes) :

SOUS-LATTICIEL (H)

1. Initialisation : $\Lambda \leftarrow (\{u_o\}, \emptyset)$; $B \leftarrow \{u_o\}$

2. Itérer tant que $[B \neq \emptyset]$

2.1. Prendre et supprimer de B un sommet u

2.2. Choisir dans $\mathcal{E}$ un connecteur $\Gamma_j(u)$

2.3. Itérer sur $\{v \in \Gamma_j(u)\}$
Si ($v \notin \Lambda$ et $v \notin W$), alors mettre v dans B
$\Lambda \leftarrow \Lambda \cup (\{v\}, (u,v))$
Fin itération 2.

3. Retourner (Λ)

On peut être intéressé non pas par l'obtention d'un sous-latticiel solution quelconque, mais par l'obtention d'un SL optimal relativement à un critère coût. Comme dans le modèle des sections précédentes, on associe un coût constant, positif ou nul, à chaque règle de décision, i.e. à chaque connecteur, et on suppose les coûts additifs.

Une définition récursive du coût d'un SL de H, Λ de racine u_o est :

$$\Psi(\Lambda) = K(u_o, j) + \sum_{v \in \Gamma_j(u_o)} \Psi(\Lambda(v)) \quad ; \text{ avec}$$

$\Gamma_j(u_o)$: ensemble des fils de u_o dans Λ ;

$K(u_o, j)$: coût du connecteur $\Gamma_j(u_o)$;

$\Lambda(v)$: sous-graphe de Λ engendré par v et ses descendants dans Λ .

Remarquons que le coût K(v,j) intervient dans $\Psi(\Lambda)$ autant de fois que sera appliquée la règle correspondante au connecteur $\Gamma_j(v)$ dans la solution représentée par Λ. Ce nombre de fois, noté m(v), est égal au nombre de chemins dans Λ de u_o à v. On vérifie sans difficulté que :

$$m(v) = \sum_{\text{arc } (u,v) \in \Lambda} m(u) \quad ; \text{ et } m(u_o) = 1$$

$$\Psi(\Lambda) = \sum_{v \in \Lambda} m(v)\, K(v,j)$$

Par extension, on prendra m(v) = 0 pour tout $v \notin \Lambda$.

Exemple 2.4. : pour le SL de la figure 2.2., on vérifie que :

$m(u_o) = m(u_1) = m(u_2) = m(u_7) = m(u_{14}) = m(u_{20}) = 1$;

$m(u_{13}) = m(u_{18}) = m(u_{21}) = 2$;

$m(u_{19}) = 4$; $m(u_{23}) = 5$; et $m(u_{22}) = 6$.

Le coût de ce SL pour la distribution de coût de la figure 2.3. est :

$$\Psi(\Lambda) = 5 + 4 + 4 + 2 \times 6 + 5 + 4 \times 25 + 2 \times 15 + 1 \times 15 = 175.$$

Soit $h^*(u)$ le coût d'un SL optimal de racine u, et h(u) un estimateur heuristique de $h^*(u)$. L'algorithme AO^* qui est une version modifiée de celui initialement proposé par MARTELLI[147], utilise l'estimateur h pour générer un SL optimal.

L'algorithme est le suivant :

Algorithme AO*

Données d'entrée : $H = (U, \Gamma)$, u_0, W, coûts $K \geqslant 0$, fonction h

1. Initialisation : $P' \leftarrow \{u_0\}$

2. Itérer tant que $[P' \neq \emptyset]$

2.1. Prendre un sommet u de P', le supprimer de P et P' et le mettre dans Q

2.2. $I(u) \leftarrow \emptyset$

2.3. Itérer sur $\{1 \leqslant j \leqslant \mathcal{J}(u)\}$

Itérer sur $\{v \in \Gamma_j(u) \mid v \notin P \cup Q\}$

$P \leftarrow P \cup \{u\}$; $f(v) \leftarrow h(v)$

Fin

$f_j(u) \leftarrow K(u,j) + \sum_{v \in \Gamma_j(v)} f(v)$

Fin itération 2.3.

2.4. $f(u) \leftarrow \min_j \{f_j(u)\}$; $\hat{j}(u) \leftarrow$ indice du minimum

2.5. $A \leftarrow \Gamma_H^{-1}(u)$

2.6. Pour $\{v \in \Gamma_H^{-1}(u)\}$ faire : $I(v) \leftarrow \{j \mid u \in \Gamma_j(u)\}$

2.7. Itérer tant que $[A \neq \emptyset]$

Prendre et supprimer de A un sommet u tel que $A \cap \Gamma_H(u) = \emptyset$

Pour $\{j \in I(u)\}$ faire : $f_j(u) \leftarrow K(u,j) + \sum_{v \in \Gamma_j(u)} f(v)$.

$\hat{j}(u) \leftarrow$ indice du nouveau $\min\{f_j(u)\}$

Si $f(u) \neq f_j(u)$, alors faire :

$f(u) \leftarrow f\hat{j}(u)$

$A \leftarrow A \cup \Gamma_H^{-1}(u)$

Pour $\{v \in \Gamma^{-1}(u)\}$ faire : $I(v) \leftarrow I(v) \cup \{j \mid u \in \Gamma_j(v)\}$

$I(u) \leftarrow \emptyset$

Fin itération 2.7.

$\Lambda \leftarrow$ SOUS-LATTICIEL (H)

$P' \leftarrow \{v \in \Lambda \mid v \in P \text{ et } v \notin W\}$

Fin itération 2.

3. Fin

Données de sortie : Λ et $f(u_0)$

Cet algorithme procède en explorant un SL partiel (i.e., ne se terminant pas nécessairement par des sommets de W), lequel est obtenu en suivant à partir de u_o les connecteurs estimés les meilleurs. A tout sommet u de H rencontré par l'algorithme est associée une estimation f(u) de $h^*(u)$:

- si u est un sommet pendant : f(u) = h(u) ;
- si u est développé, on définit pour chacun de ses connecteurs $\Gamma_j(u)$ la quantité : $f_j(u) = K(u,j) + \sum_{v \in \Gamma_j(u)} f(v)$, et l'estimation de u est : $f(u) = \min_j \{ f_j(u) \}$. On note :

. $\hat{j}(u)$: indice du minimum précédent, $\Gamma_{\hat{j}}(u)$ est le meilleur connecteur issu de u à un moment donné ;

. P' : sous-ensemble de sommets pendants non terminaux du SL Λ en cours d'exploration.

AO^* comporte une itération principale (2) qui se décompose en trois phases :

1) Le développement d'un sommet pendant dans le SL partiel Λ en cours d'exploration (les performances de l'algorithme sont sensibles au choix plus ou moins judicieux de ce sommet dans P') ;

2) La mise à jour de l'estimation f et du marquage $\hat{j}$ du connecteur minimal pour <u>tous</u> les ascendants du sommet développé (A est l'ensemble des sommets à actualiser, et I(u) désigne l'ensemble des connecteurs issus de u pour lesquels fj(u) doit être mis à jour) ;

3) La définition du nouveau SL partiel Λ et de ses somments pendants $P' \subset P$ par un parcours descendant de H le long des meilleurs connecteurs $\Gamma_{\hat{j}}(u)$. Ceci est effectué par la procédure SOUS-LATTICIEL dans laquelle on fixe l'étape non déterministe par le choix de $\Gamma_{\hat{j}}(u)$ (instruction 2.2.) et on arrête le parcours descendant de H aux sommets pendants ou terminaux.

La notion de développement d'un sommet n'a pas exactement le même sens des AO* que dans A*. Dans ce dernier algorithme, un sommet u dont l'évaluation f(u) a baissé est remis pendant, et sera éventuellement redéveloppé pour rediriger la recherche à partir de u vers un autre de ses successeurs. Dans l'algorithme AO*, le développement d'un sommet u s'effectue au plus une fois, mais la mise à jour de son évaluation f(u) et le remarquage de son connecteur minimal $\Gamma_j(u)$ se feront aussi souvent qu'un nouveau descendant de u sera développé.

Notons également une autre différence importante : dans A*, tous les chemins partiels du graphe G sont directement accessibles à tout moment de l'algorithme, et les évaluations de leur coût sont connues (c'est f(u) sur les sommets pendants) ; dans AO*, un seul SL partiel est connu à la fois, et chaque itération le redéfinit et recalcule son évaluation.

Exemple 2.5. : appliquons AO* à l'hyperlatticiel de l'exemple précédent avec les distributions de coût et l'heuristique suivantes : $K(u_0,1) = 5$ $K(u_0,2) = 19$, ..., et $h(u_1) = 20$; $h(u_2) = 30$; ... (Cf. figure 2.3.)

La séquence d'itérations (développement d'un sommet, actualisation, définition de Λ et de P')effectuées par AO* sur H est donnée par le tableau suivant. On y note f(u) par la liste ordonnée des $f_j(u)$ en soulignant $f_j(u)$. La solution obtenue est indiquée sur la figure 2.3. en traits renforcés, son coût est $f(u_0) = 74 = \Psi(\Lambda)$.

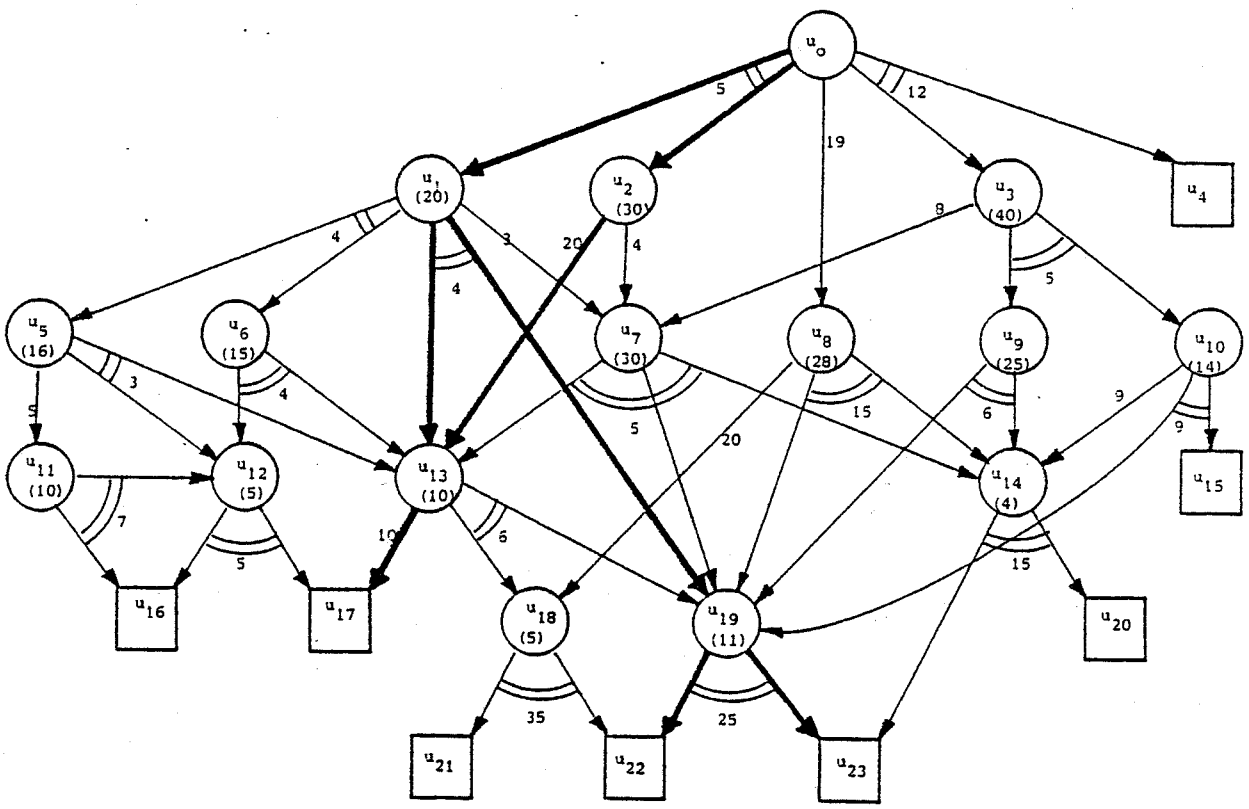


FIGURE 2.3.

u	h(u)	f(u) : $(f_1, \ldots, f_J)$	Sommets actualisés après le développement de u	P'
u_o		(55,47,52)	∅	u_8
u_8	28	(25,30)	u_o : (55,44,52)	u_{18}
u_{18}	5	(35)	u_8 : (55,30) ; u_o : (55,49,52)	u_{19} ; u_{14}
u_{19}	11	(25)	u_8 ; (55,44) ; u_o : (55,63,52)	u_3
u_3	40	(38,44)	u_o : (55,63,50)	u_7
u_7	30	(44)	u_3 : (52,44) ; u_o : (55,63,56)	u_1 ; u_2
u_1	20	(35,39,47)	u_o : (70,63,56)	u_9 ; u_{10}
u_9	25	(35)	u_3 : (52,54) ; u_o : (70,63,64)	u_{14}
u_{14}	4	(15)	u_9 : (46) ; u_8 : (55,55) ; u_7 (55) u_1 : (35,39,58) ; u_3 : (63,65) ; u_o : (70,74,75)	u_5 ; u_2 ; u_6
u_5	16	(15,18)	u_1 : (34,39,58) ; u_o : (69,74,75)	u_2 ; u_6 ; u_{11}
u_2	30	(30,59)	∅	u_6 ; u_{13} ; u_{11}
u_6	15	(19)	u_1 : (38,39,58) ; u_o : (73,74,75)	u_{13} ; u_{11} ; u_{12}
u_{13}	10	(10,66)	∅	u_{11} ; u_{12}
u_{11}	10	(12)	u_5 : (17,18) ; u_1 : (40,39,58) u_o : (74,74,75)	∅

Etablissons, à présent, l'admissibilité d'AO* :

Proposition 2.14. : Si h est minorante, alors à la fin de chacune des itérations (2.) de l'algorithme AO*, tout sommet u de H développé ($u \in Q$) vérifie : $f(u) \leq h^*(u)$; et si $\Lambda(u)$ est complet : $f(u) = h^*(u) = \Psi(\Lambda(u))$.

Preuve : Montrons, tout d'abord, que si tous les fils de u vérifient la proposition, alors u la vérifie également. Par définition (en 2.4.) ou actualisation (en 2.7.), on a :

$$f(u) = \min_j \left\{ K(u,j) + \sum_{v \in \Gamma_j(u)} f(v) \right\} \leq \min_j \left\{ K(u,j) + \sum_{v \in \Gamma_j(u)} h^*(v) \right\} = h^*(u)$$

Par ailleurs, si $\Lambda(u)$ est complet (i.e., le sous-graphe de Λ engendré par u et de ses descendants dans Λ ne comporte que des sommets développés ou terminaux), alors $\forall\, v \in \Gamma_{\hat{j}}(u)$

- ou bien v est terminal et : $f(u) = h^*(v) = \Psi(\Lambda(v)) = 0$

- ou bien $\Lambda(v)$ est complet, et par hypothèse : $f(v) = h^*(v) = \Psi(\Lambda(v))$.

D'où :

$$f(u) = K(u,\hat{j}) + \sum_{v \in \Gamma_{\hat{j}}(u)} h^*(v) = \min_j \left\{ K(u,j) + \sum_{v \in \Gamma_j(u)} h^*(v) \right\} = h^*(u) = \Psi(\Lambda(u)).$$

Montrons, à présent, la proposition par récurrence.

Base : La première itération (2.) développe

u_0 : $f(u_0) = \min_j \left\{ K(u_0,j) + \sum_{v \in \Gamma_j(u_0)} h(v) \right\} \leq \min_j \left\{ K(u_0,j) + \sum_{v \in \Gamma_j(u_0)} h^*(v) \right\} = h^*(u_0)$.

Induction : Si la proposition est vraie jusqu'à l'itération qui précède celle où u est développé, alors au moment du développement de u, pour tous ses fils v déjà existants : $f(v) \leq h^*(v)$ par induction, et pour tous ses fils nouvellement mis dans P : $f(v) = h(v) \leq h^*(v)$ par hypothèse sur h. Donc, tous les fils de u vérifient la proposition, et u la vérifie aussi d'après ce qui précède.

Il reste à établir que la proposition est encore vraie sur tous les autres sommets précédemment développés, même après l'actualisation qui suit le développement de u. Procédons de nouveau par récurrence sur la séquence des sommets actualisés, soit v le premier sommet mis à jour dans l'itération (2.7.): $f(v) = \min_j \{K(w,j) + \Sigma_{w \in \Gamma_j(v)} f(w)\}$ à la suite de cette actualisation. Pour chacun des sommets w de cette expression, ou bien f(w) n'a pas encore été modifié au cours de cette actualisation et, du fait de l'absence de cycle dans H et de l'ordre des actualisations dans (2.4.), f(w) ne pourra plus être modifié et w vérifie la proposition par induction ; ou bien w est le sommet u qui vient d'être développé et, dans ce cas, le résultat est également valide : tous les fils de v vérifient la proposition, et donc v la vérifie aussi. Un argument identique montre que pour tout fils w d'un sommet mis à jour, ou bien w n'a pas encore été modifié et ne le sera plus, ou bien w a été actualisé et vérifie la proposition par induction, ou bien w est le sommet u qui vient d'être développé. □

Proposition 2.15. : Pour un hyperlatticiel H, fini ou infini, à coûts positifs ou nuls, n'incluant pas de SL infini et de coût fini, mais possédant un sous-latticiel fini, et pour une heuristique h minorante, AO* est admissible sur H, i.e., l'algorithme s'arrête et fournit un SL optimal.

Preuve : Montrons, tout d'abord, par l'absurde que l'algorithme s'arrête nécessairement. Chaque sommet n'étant développé qu'au plus une fois, AO* ne peut itérer indéfiniment qu'en développant à chaque itération un nouveau sommet. L'absence de SL de H infini et de coût fini, implique qu'à un certain moment $f(u_o) > \Psi(\lambda)$, λ étant le SL qu'on suppose contenu dans H. Or, d'après la proposition précédente, $f(u_o) \leqslant h^*(u_o) \leqslant \Psi(\lambda)$. D'où la contradiction : AO* s'arrête nécessairement. Or, il ne peut le faire qu'avec P' = Ø. Dans ce cas, la solution Λ donnée en sortie est un SL complet et, d'après la proposition 2.14. : $f(u_o) = h^*(u_o) = \Psi(\Lambda)$. □

Si on lève la restriction faite jusqu'à présent, en admettant dans H des sommets non terminaux sans successeur, il suffira de prendre $f(u) = \infty$ lors du développement de tels sommets. Par la suite, l'actualisation ascendante des coûts redirigera les marquages $\mathfrak{f}(v)$ des connecteurs minimaux pour "condamner" l'utilisation de u dans tout SL.

On peut aussi envisager le cas d'un hyperlatticiel H fini ne contenant pas de solution. Le test d'arrêt suivant doit alors être ajouté à la fin de l'itération (2.). "Si P = ∅ et Λ contient des sommets sans successeurs, mais non terminaux, alors arrêt de l'algorithme avec absence de solution".

On a vu, dans la section précédente, qu'une heuristique monotone permet de simplifier la recherche dans un graphe. En est-il de même pour la recherche dans un hyperlatticiel ?

Ici, la définition de la monotonie de h exige que pour tout connecteur $\Gamma_j(u)$: $h(u) \leq K(u,j) + \Sigma_{v \in \Gamma_j(u)} h(v)$.

On transpose, sans difficulté, la propriété d'une heuristique monotone établie précédemment : si h est monotone et coïncidante, h est également minorante. On peut également établir :

Proposition 2.16. : Si h est monotone et coïncidante, alors pour tout sommet u développé par AO^*, les valeurs successives de f(u) forment une séquence non décroissante.

Preuve : immédiate par récurrence sur les mises à jour successives de f(u). □

On peut déduire de ce résultat que si l'algorithme développe un sommet u, qui est fils (entre autres) de v le long d'un connecteur qui n'est pas minimal (i.e., $u \in \Gamma_j(v)$ et $j \neq \hat{j}(v)$), alors durant l'actualisation, $f_j(v)$ ne peut qu'augmenter, et $f(v) = \min_j \{f_j(v)\} = f_{\hat{j}}(v)$ restera inchangé, tout comme $\hat{j}(v)$. Il semble donc inutile de mettre à jour un tel sommet v, et MARTELLI (147), ainsi que NILSSON (171) proposent de restreindre l'actualisation et l'ensemble A aux seuls pères de u dont u est fils le long du connecteur minimal (i.e., $A \leftarrow \{v \in \Gamma_H^{-1}(u) \mid u \in \Gamma_{\hat{j}}(v)\}$).

Cette restriction réduit le nombre de sommets de A, et donc le nombre d'itérations (2.7.). Apparemment, elle conduit à une amélioration des performances de l'algorithme. Mais, ceci n'est toujours pas le cas, et suivant le type de

structures de données utilisées dans une implémentation effective, elle pourraît même produire le résultat inverse.

En effet, une actualisation partielle de H conduit à rechercher pour un sommet u tous ses pères $v \in \Gamma_H^{-1}(u)$, puis à déterminer parmi eux ceux dont u est fils sur leur connecteur minimal ; ce que ne fait pas une actualisation complète. Par ailleurs, dans le cas d'une actualisation complète, il suffit de reporter systématiquement sur tous les ascendants de u l'incrément de coût $[f(u) - h(u)]$ pour l'ensemble $I(v)$ des connecteurs concernés. Dans une actualisation partielle, on ne sait pas si une estimation $f(w)$ a déjà été reportée sur les pères de w, et dans le doute, on doit faire comme si elle ne l'a pas été. Ceci nécessitera, pour chaque sommet v à actualiser, de rechercher tous ses fils w et leur estimation $f(w)$, et de refaire les calculs des $f_j(v)=K(v,j) + \Sigma f(w)$.

Un autre avantage de l'actualisation complète est qu'elle met à jour aussi bien les sommets sur les connecteurs minimaux, que ceux qui ne le sont pas ; ce qui permet d'améliorer l'estimation sur les composantes f_j, comme sur les composantes $f_{j\neq j}$. Ceci sera mis à profit par l'algorithme AO_ε pour réduire globalement la complexité de AO^*.

°0°

2.5. RECHERCHE HEURISTIQUE ϵ-ADMISSIBLE DANS UN HYPERGRAPHE : AOε

L'idée de base est la même que celle guidant A_ε : poursuivre l'exploration et le "développement en profondeur" d'un même SL Λ tant que Λ reste acceptable, i.e. tant que Λ est le meilleur SL de H, ou ne s'écarte pas de plus de $(1+\varepsilon)$ fois une estimation de meilleur SL.

Cependant, on doit faire face ici à une difficulté importante : alors que A^* dispose en permanence de la liste de tous les chemins pendants et d'une estimation de leurs coûts (tout sommet de P représente un tel chemin), dans AO^*, on ne connaît à tout moment qu'un seul SL, celui en cours de développement, et l'accès à un autre SL nécessite une actualisation ascendante de tout H, puis un parcours descendant.

Deux conditions seront nécessaires pour la mise en oeuvre d'une recherche ϵ-admissible :

- il faut être en mesure, après le développement d'un sommet u, de mettre à jour l'estimation du coût global du SL Λ en cours d'exploration, et cela sans actualisation ascendante de tout H ;

- il faut disposer, en permanence, d'un estimateur du coût du deuxième meilleur SL de H, lequel deviendra éventuellement le meilleur à la suite du développement de sommets sur Λ, et sera utilisé pour juger de l'acceptabilité de Λ. Notons que le développement d'un sommet de Λ peut avoir une incidence également sur ce deuxième SL ; il faut donc que l'estimation de son coût soit également mise à jour, si nécessaire (et sans actualisation de tout H).

Comme pour A_ϵ, un SL sera dit acceptable si l'estimation de son coût est inférieure ou égale à $(1+\varepsilon)\ h^*(u_o)$. Ne connaissant pas $h^*(u_o)$, on utilisera le plus grand des minorants connu à un moment, i.e., le maximum des estimations de coût du meilleur SL courant de H.

On reprend les notations de la section précédente : ensembles P et Q ; sous-latticiel partiel courant Λ et ses sommets pendants P' ; $\Lambda(u)$: sous-graphe de Λ engendré par u et ses descendants dans Λ ; estimation $f_j(u)$ et $f(u) = \min_j \{f_j(u)\}$; connecteur minimal $\Gamma_{\hat{j}}(u)$; et m(u):nombre de chemins distincts dans Λ de u_o à u . On définit en plus :

- $f'(u) = \min\{f_j(u) \mid j \neq \hat{j}(u)\}$, i.e., estimation du coût le long du deuxième meilleur connecteur issu de u (le meilleur étant $f_{\hat{j}}(u)$). Si u ne possède qu'un seul connecteur $f'(u) = \infty$ (tout comme $f(u) = \infty$ si u n'a pas de connecteur) ;

- F : estimateur du coût cumulé du SL Λ en cours d'exploitation ;

- F': estimateur du coût du deuxième meilleur SL de H ; lequel deviendra éventuellement le meilleur après développement de sommets dans Λ ;

- λ : meilleure solution complète connue à un moment donné (λ n'était pas acceptable au moment où il a été trouvé, mais peut éventuellement le devenir plus tard) ;

- Y : coût de λ ;

- y : seuil d'acceptabilité, correspond au plus grand estimateur minorant de $h^*(u_o)$.

Par ailleurs, les relations d'adjacence dans H (Γ_H, Γ_H^{-1} et $\hat{\Gamma}_H, \hat{\Gamma}_H^{-1}$) sont utilisées dans l'algorithme relativement à l'état instantané de H, i.e., H restreint aux sommets déjà rencontrés (ceux de P U Q).

Pour simplifier la présentation d'AO_ϵ , on fera appel aux procédures et fonctions: INITIALISER, CHOIX, DEVELOPPER, ACTUALISER, SOUS-LATTICIEL et MISE-A-JOUR. L'algorithme est le suivant.

Algorithme AO_ϵ

Données d'entrée : $H = (U,\mathcal{E})$, u_o, W, coûts $K \geq 0$, fonction h, paramètre $\epsilon \geq 0$

1. INITIALISER

2. Itérer tant que $[P' \neq \emptyset$ et $Y > (1+\epsilon)\, y]$

2.1. $u \leftarrow$ CHOIX (P')

2.2. DEVELOPPER (u)

2.3. Si $[F \leq F'$ et $f(u) \geq h(u)]$, alors faire :

2.3.1. $F' \leftarrow \min \{F' ; F + \min \{ m(v).(f'(v) - f(v)) \mid v \in \Lambda(u) \cap Q\}\}$

2.3.2. $y \leftarrow \max \{y ; F\}$

2.4. Sinon faire :

2.4.1. Si $f(u) \geq h(u)$, alors $y \leftarrow \max \{y ; F'\}$

2.4.2. Itérer tant que $[F \leq (1+\epsilon)\, y$ et $P' \neq 0]$

$u \leftarrow$ CHOIX (P')

DEVELOPPER (u)

Fin itération 2.4.2

2.4.3. Si $[F > (1+\epsilon) y]$, alors faire :

2.4.3.1. Si $[P' = \emptyset$ et $Y > F]$, alors faire :

$\lambda \leftarrow$ SOUS-LATTICIEL (H)

$y \leftarrow F$

2.4.3.2. ACTUALISER

2.4.3.3. $\Lambda \leftarrow$ SOUS-LATTICIEL (H)

2.4.3.4. $P' \leftarrow \{v \in \Lambda \mid v \in P$ et $v \notin W\}$

2.4.3.5. $F \leftarrow f(u_o)$

2.4.3.6. $F' \leftarrow f(u_o) + \min \{ m(v).(f'(v) - f(v)) \mid v \in \Lambda \cap Q\}$

2.4.3.7. $y \leftarrow \max\{y ; F\}$

2.5. Fin itération 2.

3. Fin.

Données de sortie : Si $[P' \neq \emptyset$ ou $Y < F]$, alors sortir : λ, Y et $\epsilon' \leftarrow (Y-y)/y$

Sinon sortir : $\Lambda \leftarrow$ SOUS-LATTICIEL(H), F, et $\epsilon' \leftarrow (F-y)/y$

La procédure INITIALISER développe la racine u_o de H en créant tous ses fils, détermine les estimations $f(u_o)$ et $f'(u_o)$, définit m(v) pour les fils de u_o le long du meilleur connecteur, et initialise P, Q, P', A, ainsi que F, F', Y et y.

INITIALISER

1. $P \leftarrow \emptyset$; $Q \leftarrow \{u_o\}$; $A \leftarrow \emptyset$; $Y \leftarrow \infty$; $m(u_o) \leftarrow 1$
2. Itérer sur $\{1 \leq j \leq J(u_o)\}$
   Itérer sur $\{u \in \Gamma_j(u_o)\}$
   $f(v) \leftarrow h(v)$
   $m(v) \leftarrow 0$
   $I(v) \leftarrow \emptyset$
   $P \leftarrow P \cup \{u\}$
   Fin
   $f_j(u_o) \leftarrow K(u_o, j) + \sum_{v \in \Gamma_j(u_o)} f(v)$,
   Fin itération 2.
3. $F \leftarrow f(u_o) \leftarrow \min_j \{f_j(u_o)\}$; $\hat{j}(u_o) \leftarrow$ indice du minimum
4. $F' \leftarrow f'(u_o) \leftarrow \min_{j \neq \hat{j}(u_o)} \{f_j(u_o)\}$
5. $P' \leftarrow \{v \in \Gamma_{\hat{j}}(u_o) \mid v \notin W\}$
6. $y \leftarrow \max \{h(u_o), F\}$
7. Pour $\{v \in P'\}$, faire : $m(v) \leftarrow 1$
8. Fin.

Après l'initialisation, l'algorithme AO_ε comporte quatre phases :

1) Développement d'un sommet pendant sur Λ meilleur SL de H. L'algorithme itèrera (itération 2.) en poursuivant le développement d'autres sommets de Λ , et cela tant qu'on peut garantir que Λ est le meilleur SL de H courant. Cette phase comporte également une surveillance et, éventuellement, une modification de l'estimateur F' du coût du deuxième meilleur SL de H, ainsi que du plus grand minorant de $h^*(u_o)$;

2) Poursuite du développement de sommets pendants sur le SL Λ qui n'est plus nécessairement le meilleur sous-latticiel, mais qui reste acceptable relativement au meilleur.

Cette phase a lieu lorsque $F \leq (1+\varepsilon)y$ et se trouve localisée dans l'itération (2.4.2.). Elle représente une "descente en profondeur" dans la recherche, et peut conduire à un SL complet, qui, s'il est acceptable, produit l'arrêt d'AO$_\varepsilon$ ($F \leq (1+\varepsilon)y$ et $P' = \emptyset$), et s'il ne l'est pas, sera éventuellement mémorisé comme le meilleur SL complet connu jusqu'alors (test 2.4.3.1, SL λ de coût Y) ;

3) Exécution de toutes les mises à jour reportées jusqu'à ce point. Ceci est réalisé par la procédure ACTUALISER qui procède d'une manière ascendante dans H, en actualisant le connecteur minimal $\hat{j}$ et les estimations f et f' de tous les sommets dont un de leurs fils a changé d'estimateur f.

Remarquons que si le connecteur minimal $\hat{j}(u)$ a changé, mais que la valeur du minimum de $\{f_j(u)\}$ est restée la même, avant et après mise à jour, alors l'actualisation s'arrête à u sans affecter ses ascendants ;

4) Parcours descendant de H par la procédure SOUS-LATTICIEL le long des connecteurs minimaux pour la définition du nouveau Λ, de l'ensemble P' de ses sommets pendants, des estimations F, F' et y, et des facteurs m(v) sur tous les sommets de Λ, ceux ne figurant pas dans Λ étant mis à zéro (une implémentation regroupera, bien entendu, en une seule procédure les 5 instructions 2.4.3.3. - 2.4.3.7.).

Le choix dans P', ensemble des sommets pendants de Λ du sommet particulier u qui sera développé, est effectué par la fonction CHOIX (P'). Pour la logique de l'algorithme, cette fonction peut être arbitraire, mais les performances d'AO$_\varepsilon$ seront sensibles (et à un plus grand degré que celles d'AO*) à la pertinence du choix.

Le sommet sélectionné par CHOIX est alors fourni comme argument à la procédure DEVELOPPER qui génère ses successeurs, détermine $\hat{j}(u)$, $f(u)$, et $f'(u)$, puis met à jour F, P' et si nécessaire, l'ensemble A et les facteurs $m(v)$ des descendants de u le long des connecteurs $\hat{j}(u)$.

DEVELOPPER (u)

1. $Q \leftarrow Q \cup \{u\}$; $P \leftarrow P - \{u\}$
2. Itérer sur $\{ 1 \leq j \leq J(u) \}$
   Itérer sur $\{ v \in \Gamma_j(u) \mid v \notin P \cup Q \}$
   $f(v) \leftarrow h(v)$
   $I(v) \leftarrow \emptyset$
   $m(v) \leftarrow 0$
   $P \leftarrow P \cup \{v\}$
   fin
   $f_j(u) \leftarrow K(u,j) + \sum_{v \in \Gamma_j(u)} f(u)$
   fin itération 2.
3. $f(u) \leftarrow \min_j \{f_j(u)\}$; $\hat{j}(u) \leftarrow$ indice du minimum
4. $f'(u) \leftarrow \min \{f_j(u) \mid j \neq \hat{j}(u)\}$
5. $F \leftarrow F + m(u)\,[f(u) - h(u)]$
6. Pour $\{v \in \Gamma_{\hat{j}}(u)\}$ faire : MISE-A-JOUR (v, m(u))
7. $P' \leftarrow (P' - \{u\}) \cup \{v \in \Gamma_{\hat{j}}(u) \mid v \notin W\}$
8. Si $(f(u) \neq h(u))$, alors faire :
   $A \leftarrow A \cup \Gamma_H^{-1}(u)$
   Pour $\{w \in \Gamma_H^{-1}(u)\}$ faire : $I(w) \leftarrow I(w) \cup \{j \mid u \in \Gamma_j(v)\}$
9. Fin.

La mise à jour des facteurs $m(u)$ des descendants de u le long des connecteurs $\hat{j}$ se fait très simplement par la procédure récursive suivante :

MISE-A-JOUR (v,y)

1. Si ($v \notin W$), alors $m(v) \leftarrow m(v) + y$
2. Si ($v \notin P'$), alors Pour $\{ w \in \Gamma_{\hat{j}}(v) \}$ faire : MISE-A-JOUR (w,y)
3. Fin.

La procédure assurant l'actualisation ascendante de H est la suivante :

ACTUALISATION

1. Itérer tant que $[A \neq 0]$
- 1.1. Prendre et supprimer de A un sommet u tel que $A \cap \hat{\Gamma}_H^{-1}(u) = \phi$
- 1.2. Pour $\{ j \in I(u) \}$ faire : $f_j(u) \leftarrow K(u,j) + \sum_{v \in \Gamma_j(u)} f(v)$
- 1.3. Chercher le min $\{ f_j(u) \}$ et soit $\hat{j}(u)$ son indice
- 1.4. $f'(u) \leftarrow \min_j \{ f_j(u) \mid j \neq \hat{j}(u) \}$
- 1.5. $I(u) \leftarrow \emptyset$
- 1.6. Si $f(u) \neq f_{\hat{j}}(u)$, alors faire :
  $A \leftarrow A \cup \Gamma_H^{-1}(u)$
  $f(u) \leftarrow f_{\hat{j}}(u)$
  Pour $\{ v \in \Gamma^{-1}(u) \}$ faire $I(v) \leftarrow I(v) \cup \{ j \mid u \in \Gamma_j(v) \}$
2. Fin.

Notons, finalement, qu'après une actualisation, le parcours descendant de H devra calculer correctement les facteurs $m(v)$: mise à zéro pour les sommets ne figurant pas dans le nouveau Λ ; et $m(v) = \Sigma m(u)$: somme sur les arcs (u,v) de Λ. Ce calcul est facilement intégré à la procédure SOUS-LATTICIEL :

- $\forall v \in P \cup Q$: $m(v) \leftarrow 0$ initialement ; puis
- $m(v) \leftarrow m(v) + m(u)$ dans la boucle (2.3).

La preuve de l'ε- admissibilité d'AO$_\varepsilon$ procèdera en 4 étapes qui analyseront successivement la phase 3 d'actualisation, la phase 1 de développement de sommets sur le meilleur SL de H, la phase 2 de "descente en profondeur" dans un SL acceptable, puis les conditions d'arrêt de l'algorithme.

Les hypothèses des propositions 2.17 à 2.20 sont les suivantes :

h est une heuristique minorante, H est un hyperlatticiel à coûts positifs ou nuls, n'incluant pas de sous-latticiel infini à coût fini, et contenant au moins un SL fini.

Proposition 2.17. : A la suite d'une actualisation de H et d'un parcours descendant le long des connecteurs $\hat{j}(u)$, les redéfinitions de Λ, F et F' (2.4.2.3. - 2.4.2.6.) vérifient :

a) F est un estimateur minorant et coïncidant du coût de Λ; meilleur SL de H, i.e., $F \leq h^*(u_o)$, et si Λ est complet $F = \Psi(\Lambda) = h^*(u_o)$;

b) F' est un estimateur minorant du coût du deuxième meilleur SL de H.

Preuve : Au moment où commence une mise à jour, l'ensemble A contient tous les sommets qui ont eu un fils dont l'estimation f a varié.

L'actualisation procède d'une manière identique à celle d'AO*, sauf que :

- pour garantir une actualisation effectivement ascendante, on vérifie que $A \cap \hat{\Gamma}_H(u)$ (au lieu de $A \cap \Gamma_H(u) = \phi$ seulement pour AO*) ; ceci étant dû au fait que A reçoit des sommets à divers moments ;

- pour chaque sommet u, on retient, en plus de l'estimateur f(u) du coût le long du meilleur connecteur, l'estimation f'(u) du coût du deuxième meilleur (la variation de f'(u) seule n'entraînant pas d'actualisation pour les ascendants de u). D'où :

a) Les propriétés de f(u), après une actualisation dans A0* (proposition 2.13.), restent valables ici : $f(u) \leq h^*(u)$ et $f(u) = \psi(\Lambda(u)) = h^*(u)$ si $\Lambda(u)$ est complet. Ceci est en particulier vrai pour u_0, et donc $F = f(u_0)$ est un estimateur minorant et coïncidant de Λ, meilleur SL de H à ce moment.

b) Pour établir la deuxième conclusion, remarquons que la définition de F après une actualisation est :

$$F = f(u_0) = \sum_{v \in \Lambda - P'} m(v).K(v,\jmath) + \sum_{v \in P'} m(v)\, h(v),$$

ce qu'on peut écrire plus simplement :

$$f(u_0) = \sum_{v \in \Lambda} m(v).\ K(v), \qquad \text{(i)}$$

avec $K(v) = K(v,\jmath)$ si v est développé, et $K(v) = h(v)$ si v est pendant.

Montrons alors que pour tout sommet u de Λ, on peut mettre $f(u_0)$ sous la forme :

$$f(u_0) = \sum_{v \in (\Lambda - \Lambda(u))} m(v/u).K(v) + m(u).f(u). \qquad \text{(ii)}$$

$\Lambda(u)$: sous-graphe engendré par u et ses descendants dans Λ, son complément $(\Lambda - \Lambda(u))$ est obtenu en enlevant de Λ le sommet u et tous les arcs qui en sont issus, puis en recommençant récursivement à partir de tous les sommets sans père.

m(v/u) : nombre de chemin de u_0 à v, ne passant pas par u.

La relation (i) n'est pas valable uniquement pour u_o : elle se généralise à tout sommet de Λ, en particulier u :
$f(u) = \sum_{v \in \Lambda(u)} m(u,v) \cdot K(v)$.

où m(u,v) est le nombre de chemins distincts dans $\Lambda(u)$ de u à v.

Par ailleurs, les chemins de u_o à v se partitionnent en ceux ne passant pas par u et ceux passant par u :

$m(v) = m(v/u) + m(u)\ m(u,v)$; ce qui se réduit à
$m(v) = m(v/u)$, si v n'appartient pas à $\Lambda(u)$. D'où :
$f(u_o) = \sum_{v \in \Lambda} m(v) \cdot K(v)$

$$f(u_o) = \sum_{v \in \Lambda \text{ et } v \notin \Lambda(u)} m(v/u)K(v) + \sum_{v \in \Lambda(u)} (mv/u) + m(u)m(u,v))K(v)$$

$$f(u_o) = \sum_{v \in (\Lambda - \Lambda(u))} m(v/u)K(v) + m(u)\sum_{v \in \Lambda(u)} m(u,v)K(v)$$

$$f(u_o) = \sum_{v \in (\Lambda - \Lambda(u))} m(v/u)K(v) + m(u)f(u)$$

Estimons, à partir de là, un minorant du coût d'un SL Λ' obtenu à partir de Λ en remplaçant le connecteur $\hat{j}(u)$, issu d'un sommet u quelconque, par un autre connecteur $j(u) \neq \hat{j}(u)$. Λ' est alors composé de $(\Lambda - \Lambda(u))$ et de tous les descendants de u le long de $\Gamma_j(u)$. D'où :

$$\Psi(\Lambda') \geqslant \sum_{v \in (\Lambda - \Lambda(u))} m(v/u)k(v) + m(u)f_j(u)$$

$$\Psi(\Lambda') \geqslant f(u_o) + m(u)\,(f_j(u) - f(u)).$$

Or, $f'(u) = \min \left\{ f_j(u) \mid j \neq \hat{j}(u) \right\}$; et

$F' = f(u_o) + \min \left\{ m(u)\,(f'(u) - f(u)) \mid u \in \Lambda \cap Q \right\}$. D'où

$\Psi(\Lambda') \geqslant F'$.

L'inégalité reste, bien entendu, valide pour tout autre Λ'' qui diffèrerait de Λ par plus d'un connecteur. □

Proposition 2.18. : Si, à la suite d'une séquence de développements de sommets sur Λ, sans actualisation de H, on a tout au long de cette séquence: $F \leqslant F'$ et $f(u) \geqslant h(u)$, alors les mises à jour de F (DEVELOPPER 5.) et de F' (AO_ε 2.3.1.) vérifient les conclusions (a) et (b) de la proposition précédente.

Preuve : par récurrence sur la séquence de sommets développés. La base de la récurrence analysera l'algorithme avant et après le développement du premier sommet de la séquence. Soit u ce sommet. On considèrera trois états successifs 0, 1 et 2 :

- (0) : état de l'algorithme en fin d'itération (2.) lorsqu'une actualisation de H a été effectuée, juste avant le début de la séquence de développements ;

- (1) : état de l'algorithme en fin d'itération (2) après le développement de u. Premier sommet de la séquence ;

- (2) : état fictif qu'aurait eu l'algorithme en fin d'itération (2.) si juste après le développement de u, il procédait à une autre actualisation de H (ce qu'il ne fait pas, du fait de l'hypothèse $F \leqslant F'$ et $f(u) \geqslant h(u)$).

On indicera les données par 0, 1 ou 2 pour désigner leur valeur respective dans chacun de ces états.

a) Pour établir la première conclusion, revenons à la définition de F :

$$F_1 = F_0 + m_0(u) \times (f_1(u) - h(u)) \text{ ; et}$$

$$F_0 = \sum_{v \in (\Lambda - \Lambda_0(u))} m_0(v/u) \times K(v) + m_0(u) \times f_0(u)$$

Par ailleurs, le développement de u ne modifie dans Λ_o que les descendants de u, donc :

$$\Lambda_o - \Lambda_o(u) = \Lambda_1 - \Lambda_1(u) \quad ;$$

$m_0(u) = m_1(u)$; et $m_0(v/u) = m_1(v/u)$

De plus, u est pendant dans l'état (0) : $f_o(u) = h(u)$.
Doù : $F_1 = \sum_{v \in (\Lambda_1 - \Lambda_1(u))} m_1(v/u)\ K(v) + m_1(u)\ f_1(u)$

On retrouve pour F_1 l'expression (ii) de $f(u_o)$, et donc F_1 est un estimateur minorant et coïncidant de Λ_1. Montrons alors que Λ_1 est le meilleur SL de H à ce moment. Admettons le contraire : une actualisation de H après le développement de u aurait alors changé au moins un des connecteurs $j(v)$ pour $v \in (\Lambda_1 - \Lambda_1(u))$ (ce sont les seuls sommets de Λ_1 à être actualisés), et aurait conduit à un SL Λ_2 meilleur que Λ_1 : $F_2 \leqslant F_1$.

Par ailleurs, tout changement d'un connecteur dans Λ_o conduit à un SL de coût supérieur ou égal à F_o' (proposition précédente), et du fait de: $f(u) \geqslant h(u)$, le développement de u ne peut faire baisser le coût de ce SL : $F'_o \leqslant F_2$.

De plus, par définition :
$F'_1 = \min\left\{F'_o\ ;\ F_1 + \min\left\{m_1(v)(f'_1(v) - f(u)) \mid v \in \Lambda_1(u) \cap Q_1\right\}\right\}$;
et: $F'_1 \leqslant F'_o \leqslant F_2 \leqslant F_1$;

Or, par hypothèse, $F_1 \leqslant F'_1$; d'où la contradiction.

Donc, la deuxième actualisation de H n'aurait rien modifié à Λ_1 : $\Lambda_1 = \Lambda_2$, $F_1 = F_2$; et la conclusion (a) de la proposition précédente, valide pour Λ_2 et F_2 est aussi valide pour Λ_1 et F_1.

b) La deuxième conclusion sera également établie en montrant qu'une actualisation après le développement de u ne peut conduire à une estimation F'_2 meilleur que F'_1. Par définition :

$$F'_2 = F_2 + \min \{ m_2(v) (f'_2(v) - f_2(v) \mid v \in \Lambda_2 \cap Q_2 \}$$

Entre l'état (0) et l'état (2), seul u a été développé ; se rajoute donc à Λ_0 le sommet u et ses descendants, d'où :

$$\Lambda_2 \cap Q_2 = \Lambda_1 \cap Q_1 = (\Lambda_0 \cap Q_0) \cup (\Lambda_1(u) \cap Q_1) \quad ; \text{ et}$$

$$F'_2 = F_2 + \min \{ \min\{m_2(v)(f'_2(u) - f_2(v)) \mid \Lambda_0 \cap Q_0\} ; \min\{m_2(v)(f'_2(u) - f_2(v)) \mid \Lambda_1 \cap Q_1\}\}$$

en tenant compte de :

$F_2 = F_1 = F_0 + m_0(u) (f_1(u) - h(u))$, on obtient :

$$F'_2 = \min \begin{cases} F_0 + \min \{ m_0(u) \times [f_1(u) - h(u)] + m_2(v) \times [f'_2(v) - f_2(v)] \mid v \in \Lambda_0 \cap Q_0 \} \\ F_1 + \min \{ m_2(v) (f'_2(v) - f_2(v)) \mid \Lambda_1(u) \cap Q_1 \} \end{cases}$$

Analysons le premier terme de ce min suivant que v est ou n'st pas un descendant de u dans :

- Si $v \in (\Lambda_1 - \Lambda_1(u))$: alors $m_2(v) = m_0(v)$;

$f_2(v) = f_0(v) + m_0(v,u) (f_1(u) - h(u))$;

$f'_2(v) \geq f'_1(v) = f'_0(v)$ car le développement de u ne peut reporter sur $f'_2(v)$ qu'une quantité $k(f_1(u) - h(u) \geq 0$;

$m_0(u) = m_0(u/v) + m_0(v) \times m_0(v,u) \geq m_0(v) \times m_0(v,u)$. D'où

$m_2(v) f_2(v) \leq m_2(v) f_0(v) + m_0(u) \times (f_1(u) - h(u))$, et

$m_2(v)(f'_2(v) - f_2(v)) + m_0(u) \times (f_1(u) - h(u) \geq m_0(v) \times (f'_0(v) - f_0(v))$; (iii)

- Si $v \in \Lambda_1(u)$: le développement de u ne modifie pas $f_o(v)$ et $f'_o(v)$, par contre : $m_1(v) = m_o(v) + m_1(u,v)$, d'où : $m_2(v)(f'_2(v) - f_2(v)) \geq m_o(v)\,(f'_o(v) - f_o(v))$; et $f_1(u) \geq h(u)$, on retrouve l'inégalité (iii).

Dans tous les cas, on peut donc minorer :

$$F_o + \min\left\{ m_o(u)\,(f_1(u) - h(u)) + m_2(v)(f'_2(v) - f_2(v) \mid \Lambda_o \cap Q_o \right\}$$
$$\geq F_o + \min\left\{ m_o(v)\,(f_2(v) - f_2(u)) \mid \Lambda_o \cap Q_o \right\} = F'_o$$

d'où :

$$F'_2 \geq \min\left\{ F'_o \,;\, F_1 + \min\left\{ m_2(v)(f'_2(v) - f_2(v)) \mid \Lambda_1(u) \cap Q_1 \right\} \right\} = F'_1$$

Or, F'_2 vérifie la conclusion (b) d'après (2.17.), donc son minorant F'_1 la vérifie aussi.

On terminera la preuve en notant que l'étape d'induction est similaire à la base de la récurrence. Il suffit de reprendre les mêmes arguments en adoptant pour état (0), l'état (2) du développement précédent, et en remarquant que tous les sommets de la séquence de développements dont l'estimation est passée de h à $f \geq h$, interviennent dans A et dans F. □

Remarques :

- L'hypothèse ($f(u) \geq h(u)$) n'est pas nécessaire pour établir que F_1 est un estimateur minorant et coïncidant du coût de Λ, mais elle intervient pour montrer que Λ est le meilleur SL de H à ce moment. En effet, si $f(u) < h(u)$, le coût de Λ baissera exactement de $m(u)\,(f(u) - h(u))$; mais d'autres SL de H, où le nombre de chemins de u_o à u est supérieur à m(u), peuvent devenir meilleurs que Λ. Pour la même raison, lorsque $f(u) < h(u)$, on ne sait pas quelle est l'estimation du coût du deuxième meilleur SL de H.

- On suppose implicitement dans la preuve précédente qu'en cas d'ex-aequo entre plusieurs f_j minimaux, l'instruction (ACTUALISER 1.3.) privilegie l'absence de changement de marquage de $\hat{j}(u)$: on garde le connecteur précédent s'il n'y en a pas de strictement meilleur.

Proposition 2.19. : Tout SL Λ, sur lequel AO_ε développe un sommet, était acceptable avant ce développement.

Preuve : Montrons d'abord, par récurrence, que y est un estimateur minorant de $h^*(u_o)$. Initialement :

$$y = \max\left\{ h(u_o) \; ; \; \min_j \left\{ K(u_o, j) + \Sigma_{\Gamma_j(u_o)} h(v) \right\} \right\} \leq h^*(u_o)$$

car h est minorante.

Par la suite, y ne peut être modifié que de trois façons :

- avec $y = \max\{y, F\}$ après une actualisation (en 2.4.3.7.) ; et dans ce cas (d'après 2.17.a) : $F \leq h^*(u_o)$;

- avec $y = \max\{y, F\}$ au cours d'une séquence de développement de sommets au long de laquelle $F \leq F'$ et $f(u) \geq h(u)$; et on retrouve $F \leq h^*(u_o)$ d'après (2.18.a) ;

- avec $y = \max\{y, F'\}$ après le développement d'un sommet u tel que $f(u) \geq h(u)$ et $F > F'$ (en (2.4.1.). Avant le développement de u, F' était une estimation minorante du coût du deuxième meilleur SL de H (2.17.b ou 2.18.b). Une actualisation (fictive), après le développement de u, ne peut qu'augmenter le coût de tous les SL de H où figure u (en particulier de Λ) d'une quantité $k(f(u)-h(u)) \geq 0$. Le meilleur SL de H aura alors au mieux F' comme estimation minorante : $F' \leq h^*(u_o)$.

Le reste de la preuve est alors simple. Le développement d'un sommet s'effectue :

- soit en (2.2.): u est pendant sur le meilleur SL de H, qui est donc acceptable ;

- soit en (2.4.2.2.) : F (mis à jour) est une estimation minorante et coïncidante du coût de Λ qui vérifie :

  $F \leq (1+\varepsilon)y \leq (1+\varepsilon)\ h^*(u_o)$; et donc Λ est acceptable. □

Proposition 2.20. : Pour tout $\varepsilon \geq 0$, AO_ε est ε'- admissible avec $\varepsilon' \leq \varepsilon$.

Preuve : La preuve de l'arrêt d'AO^* se transpose directement à AO_ε. Il suffit de remplacer les inégalités :

$f(u_o) > \Psi(\Lambda)$ par $F > (1+\varepsilon)\ \Psi(\Lambda)$; et

$f(u_o) \leq h^*(u_o) \leq \Psi(\Lambda)$ par $F \leq (1+\varepsilon)\ h^*(u_o) \leq (1+\varepsilon)\Psi(\Lambda)$.

De plus, lorsque AO_ε s'arrête, une des trois conditions suivantes est remplie :

- P' = ∅ et $F \leq F'$; à la suite d'une séquence de développements de sommets sur le meilleur SL de H. Dans ce cas, Λ est complet, $F = \Psi(\Lambda) = h^*(u_o)$; et $F \leq y \leq h^*(u_o)$ y = F et $\epsilon' = 0$; i.e. l'algorithme fournit en sortie une solution optimale et une garantie de l'optimalité exacte de cette solution.

- P' = ∅ et $y < F \leq (1+\varepsilon)y$: dans ce cas, Λ est complet et ε'-optimal:
  $\epsilon' = (F-y)/y \leq \varepsilon$ et
  $F = \Psi(\Lambda) = (1+\varepsilon')y \leq (1+\varepsilon)\ h^*(u_o)$.

- $Y \leqslant (1+\varepsilon)y$: dans ce cas, λ meilleur SL complet connu par l'algorithme, qui n'était pas acceptable au moment où il a été atteint, vient de le devenir. Si ce cas d'arrêt se produit en même temps qu'un des deux précédents ($P' = \emptyset$), la sortie de λ n'a lieu que si λ est strictement meilleur que $\mathcal{L}$ (i.e. : $Y < F$) . On a alors :

$\varepsilon' = (Y-y)/y \leqslant \varepsilon$; et

$Y = \psi(\mathcal{L}) = (1+\varepsilon')y \leqslant (1+\varepsilon)\, y \leqslant (1+\varepsilon)\, h^*(u_o)$. □

Exemple 2.6. : Appliquons AO_ε à l'hyperlatticiel de l'exemple précédent (figure 2.3.) :

- pour $\varepsilon = 0.52$: l'algorithme s'arrête après avoir développé trois sommets (u_o, u_8 et u_{18}) et sans aucune actualisation, en fournissant un SL de coût 74 (marqué en trait renforcé sur la figure 2.4.);

- pour $\varepsilon = 0.43$: AO_ε aboutit à la même solution en développant un sommet de plus (u_{19}) et en effectuant une seule actualisation (après u_{18});

- pour $\varepsilon = 0.06$: après avoir développé neuf sommets (u_o, u_8, u_{18}, u_{19}, u_3, u_7, u_1, u_9 et u_{14}) et procédé à cinq actualisations (après u_{18}, u_{19}, u_7, u_1 et u_{14}), l'algorithme s'arrête avec la même solution que précédemment, garantie à moins de 5,71 % de l'optimum. Rappelons que sur ce même exemple, AO^* a effectué 14 développements de sommets et procédé à 11 actualisations.

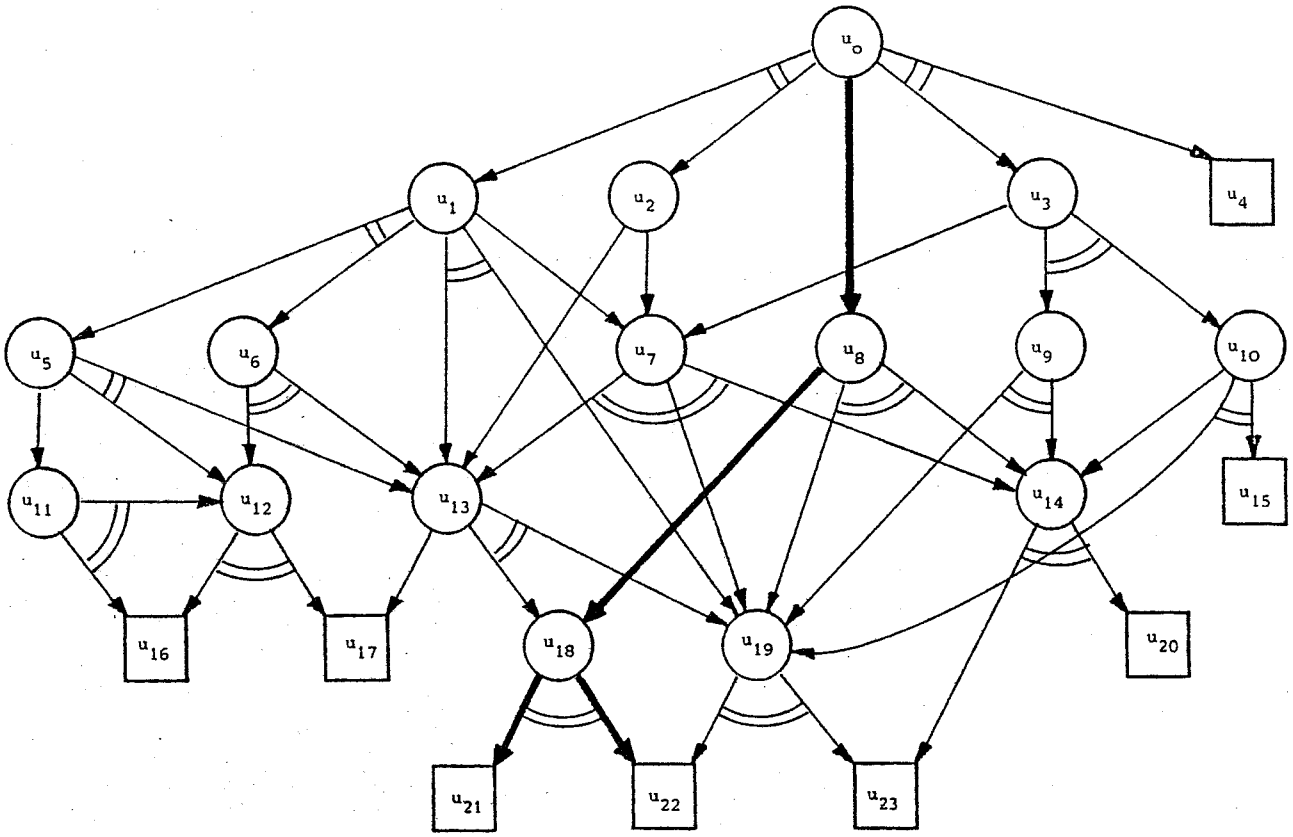


FIGURE 2.4.

Avant d'analyser certains cas particuliers, mentionnons que la formulation précédente de l'algorithme est volontairement simplifiée et ne profite pas de toutes les possibilités de l'approche ε - admissible. On peut, en effet, exploiter le principe d'acceptabilité (qui est destiné à ouvrir de nombreuses alternatives, parmi lesquelles on espère faire un choix judicieux pour la complexité), non seulement pour poursuivre l'exploration sur un SL qui n'est plus strictement le meilleur, mais aussi pour développer ce SL le long de

connecteurs qui ne sont pas les meilleurs, ou en cas de backtracking, pour revenir à un SL qui est simplement acceptable. Ainsi, dans les procédures DEVELOPPER et ACTUALISER, il n'est pas nécessaire de prendre systématiquement pour $\hat{j}(u)$ l'indice de $\min \{ f_j(u) \}$: on peut poursuivre u par tout connecteur qui conserve l'acceptabilité de Λ . Une procédure équivalente à SELECTION d'A_ε pourraît choisir dans l'ensemble :

$$\{ 1 \leq j \leq J(u) \mid \quad f_j(u) \leq ((1+\varepsilon)y - F)/m(u) + h(u) \}$$

- soit le connecteur $\Gamma_j(u)$ le plus "développé" (sur la base de la hauteur moyenne ou maximale des SL $\Lambda(v)$ pour $v \in \Gamma_j(u)$, ou d'une heuristique h_c si l'information est disponible) ;

- soit le connecteur à $f_j(u)$ minimal ;

- soit un compromis.

Monotonie de l'heuristique : Si h est monotone, les tests ($f(u) \geq h(u)$) en (2.3.) et (2.4.1.) sont inutiles et pourront être supprimés.

La phase (1) de développements de sommets sur le meilleur SL de H se poursuit alors tant que $F \leq F'$. Dès que cette inégalité n'est plus vérifiée, on pourra mettre à jour y par $y \leftarrow \max \{ y, F' \}$. On peut aussi supprimer les mises à jour de y en (2.3.2.) et en (2.4.3.7.), à condition d'en tenir compte dans la définition de ε'.

Comme pour AO*, la monotonie permet de restreindre l'actualisation aux seuls sommets v dont un fils u sur leur connecteur minimal, $u \in \Gamma_j(v)$, a changé d'estimation f(u). Mais les inconvénients de cette restriction sont encore plus importants ici : une actualisation complète permet de relever le seuil d'acceptabilité et accélère la convergence de l'algorithme.

Recherche dans un espace H dont tout SL est une arborescence

Il s'agit d'un cas particulier important pour les applications que nous développerons dans les chapitres suivants. L'espace de recherche est un hyperlatticiel H tel que tout SL de H (toute sortie de la procédure non déterministe SOUS-LATTICIEL) est une arborescence. On peut caractériser un tel espace par la condition nécessaire et suffisante suivante :

$$\forall u, v, w, j : u \in \Gamma_j(w) \text{ et } v \in \Gamma_j(w) \Rightarrow \hat{\Gamma}_H(u) \cap \hat{\Gamma}_H(v) = \phi$$

On en déduit immédiatement que $\forall u, \Lambda$: $m(u) = 1$. On peut aussi voir que même si $f(u) < h(u)$, il est possible de suivre la mise à jour de F et de F' pour déterminer quel est le meilleur SL courant de H.

Les simplifications de l'algorithme AO_ε qui en résultent sont les suivantes :

- on supprime m(u) partout où il apparaît en le remplaçant par le coefficient unitaire ; de même, on supprime la procédure MISE-A-JOUR, et l'instruction (6.) de DEVELOPPER ;

- on remplace dans AO_ε les instructions suivantes :

(2.3.) par (2.3)' : Si $(F \leq F')$ alors faire

(2.3.1.)par (2.3.1)': $F' \leftarrow \min\left\{ F' ; F' + f(u) - h(u) ; F + \min\left\{ f'(v) - f(v) \mid v \in \Lambda(u) \cap Q \right\} \right\}$

(2.4.1.)par (2.4.1)': $y \leftarrow \max\left\{ y ; F' \right\}$

On peut alors établir :

Proposition 2.21. : AO_ε modifié, avec une heuristique minorante et un espace de recherche dont tout SL est une arborescence, est ε' - admissible.

Preuve : Par différence par rapport à celle du cas général. Le résultat de (2.17) demeure inchangé. La preuve (2.18) doit être revue dans le cas $f(u) < h(u)$:

- la mise à jour de F par $F \leftarrow F + f(u) - h(u)$ est correcte même dans ce cas, i.e. F est une estimation minorante et coïncidante du coût de Λ. Montrons alors que Λ est bien le meilleur SL de H lorsque $F_1 \leqslant F'_1$. Si cela est faux, $F_2 \leqslant F_1$. Par ailleurs, tout changement d'un connecteur dans Λ_0 conduit à un SL de coût supérieur ou égal à F'_0. Or, tout le coût de tout SL de H ne peut s'améliorer du fait du développement de u de plus de $(f(u) - h(u))$, d'où :
$F_2 \geqslant \min\{F'_0 ; F'_0 + f(u) - h(u)\}$.

De plus, par définition (2.3.1.)' :
$F'_1 \leqslant \min\{F'_0 ; F'_0 + f(u) - h(u)\}$.

D'où : $F'_1 \leqslant F_2 \leqslant F_1$, ce qui contredit l'hypothèse.

- il reste à montrer que F' vérifie (b). Par définition :

$$F'_1 = \min\left\{ F'_0 ; F'_0 + f(u)-h(u) ; F_1 + \min\{f'_1(v)-f_1(v) \mid \Lambda(u) \cap Q\} \right\}$$

$$F'_2 = \min\left\{ \begin{array}{l} F_0 + f(u)-h(u) + \min\{f'_2(v)-f_2(v) \mid \Lambda_0 \cap Q_0\} ; \\ F_1 + \min\{f'_1(v) - f_1(v) \mid \Lambda(u) \cap Q\} \end{array} \right\}$$

$$F'_0 = F_0 + \min\{f'_0(v) - f_0(v) \mid \Lambda_0 \cap Q_0\} .$$

. Si $v \in \Lambda_0$, et v est un ascendant de u, alors :
$f_2(v) = f_0(v) + f_1(u) - h(u)$. Si le développement de u se reporte sur $f'_2(v)$, alors $f'_2(v) = f'_0(v) + f_1(u) - h(u)$ et

$f'_2(v) - f_2(v) = f'_0(v) - f_0(v)$; si ce développement ne se reporte pas : $f'_2(v) = f'_0(v)$, et du fait de $f_1(u) < h(u)$:
$f'_2(v) - f_2(v) = f'_0(v) - f_0(v) - (f_1(u) - h(u)) \geqslant f'_0(v) - f_0(v)$.

. si $v \in \Lambda_o$ sans être un ascendant de u, alors $f_2(v) = f_o(v)$ et $f'_2(v)$ ne peut non plus être modifié par la deuxième actualisation, car sinon Λ ne serait pas un arbre, d'où : $f'_2(v) - f_2(v) = f'_o(v) - f_o(v)$.

Donc, dans tous les cas, on pourra minorer :

$$F_o + f_1(u) - h(u) + \min\{ f'_2(v) - f_2(v) \mid \Lambda_o \cap Q_o \}$$
$$\geq F_o + f_1(u) - h(u) + \min\{f'_o(v) - f_o(v) \mid \Lambda_o \cap Q_o\}$$
$$\geq F'_o + f_1(u) - h(u)$$

d'où $F'_2 \geq F'_1$.

- Le reste de la preuve consiste à transposer (2.19.) et (2.20.) directement, en remarquant pour la mise-à-jour de y en (2.4.1.)' qu'en cette étape $F > F'$, alors qu'avant le développement de u, F était inférieur à F', par conséquence $f(u) > h(u)$ et $F' \leq h^*(u_o)$. □

Stratégie persévérante pour AO_ε

Discutons quelques parallèles entre les recherches dans un graphe et dans un hypergraphe. Pour réduire les "backtracking" dans A_ε, on avait proposé une "stratégie persévérante", destinée à élever le seuil d'acceptabilité et à permettre de poursuivre une exploration en profondeur. Peut-on étendre cette stratégie à AO_ε ?

Dans le cas d'une heuristique monotone ou d'une recherche dans un espace dont tout SL est un arbre, la réponse est théoriquement affirmative, mais pratiquement cela entraînerait un traitement algorithmique supplémentaire dont le coût rendrait le bénéfice de cette stratégie très douteux. En effet, l'absence d'accès explicite aux SL partiels de H contraint ici à :

- i) retenir le sommet $\hat{v}$ qui corespond au minimum (mis à jour) de $\{ m(v)\,(f'(v)-f(v)) \mid v \in \Lambda \cap Q \}$, i.e. le sommet pour lequel $F' = f(u_o) + m(\hat{v})*(f'(\hat{v}) - f(\hat{v}))$;

- ii) calculer, mettre à jour tout au long du développement de Λ et conserver une valeur :

$$F'' = \min\left\{f''(\hat{v}) \; ; \; f(u_o) + \min\left\{ m(v)(f'(v)-f(v)) \;\middle|\; v \in \Lambda \cap Q, v \neq \hat{v} \right\}\right\}$$

$f''(\hat{v})$ étant le troisième minimum de $\{f_j(\hat{v})\}$ si $\hat{v}$ a plus de trois connecteurs.

Au moment où Λ cesse d'être acceptable ($F > (1+\varepsilon)y$), le seuil y peut être élevé en procédant à :

1) La définition d'un SL $\Lambda'(\hat{v}) = \{$fils de $\hat{v}$ le long du connecteur $\Gamma_j(\hat{v})$ correspondant à $f'(\hat{v})$, et descendants de ces fils le long des connecteurs minimaux$\}$, et de $P'(\hat{v}) = \{$sommets pendants non terminaux dans $\Lambda'(\hat{v})\}$;

2) Le développement d'un ou plusieurs sommets w de $P'(\hat{v})$ en reportant sur F' les écarts $m'(w)(f(w) - h(w))$ et en surveillant une diminution éventuelle de F" :

$F'' \leftarrow \min\{F'' \; ; \; F' + m'(w)\,(f'(w) - f(w))\}$.

Lorsque F', mis à jour, sera supérieur à F", on aura : $F'' \leq h^*(u_o)$, et on pourra prendre $y \leftarrow \max\{y, F''\}$.

Généralisation à des hypergraphes orientés quelconques

Les algorithmes A^* et A_ε s'appliquent à des graphes à racines, quelconques, alors que AO^* et AO_ε ont été restreints à des hypergraphes sans circuit. Peut-on lever cette restriction ?

Un circuit composé de sommets n'ayant chacun qu'un seul connecteur correspond à une singularité de l'hypergraphe, en ce sens où aucune solution ne peut passer par l'un de ses sommets (elle serait condamnée à y repasser indéfiniment). Il faudrait donc donner aux algorithmes le moyen de reconnaître et d'éviter une telle singularité.

Par ailleurs, avec ou sans singularité, il faut faire en sorte que le choix des connecteurs minimaux $\hat{j}$, n'introduise pas de circuit dans le SL en cours d'exploration. Finalement, même en l'absence de cette éventualité, il faut veiller à ce que l'actualisation ascendante, en rencontrant un circuit, ne place pas l'algorithme dans une boucle infinie.

Les deux règles suivantes permettraient de prendre en compte ces objectifs :

1) Si le sommet u en cours de développement possède un fils v non pendant qui figure parmi ses ascendants, alors affecter une valeur infinie à l'estimateur correspondant de u, i.e. :
Si $\Gamma_j(u) \cap Q \cap \hat{\Gamma}_H^{-1}(u) \neq \phi$ alors, $f_j(u) \leftarrow \infty$;

2) Lors d'une mise à jour de l'ensemble A à partir d'un sommet u, ne mettre dans A que les pères v de u, tel que :
$u \in \Gamma_j(v)$ et $f_j(v) < \infty$ (l'actualisation ne faisant que reporter l'incrément de f(u) sur $f_j(v)$, serait dans ce cas inutile).

Complexité de la recherche heuristique dans un hypergraphe

Très peu de résultats sur la complexité d'AO* sont connus. Les modèles et analyses faites dans le cas d'A* ne sont malheureusement pas transposables. En particulier, on a admis que la complexité de l'algorithme A* peut se mesurer en nombre d'itérations sur sa boucle principale, i.e., en nombre total de développements de sommets, toutes les autres opérations effectuées par l'algorithme n'intervenant que pour un facteur constant sur ce nombre.

Cette hypothèse n'est pas réaliste dans le cas d'AO* : tout sommet développé peut entraîner une actualisation sur tous ses ascendants dans H, laquelle actualisation pèse lourdement sur la complexité de l'algorithme, par un facteur non constant.

On peut, par contre, admettre que le développement d'un sommet et son actualisation exigent à peu près les mêmes opérations (le calcul des $f_j(u) \leftarrow K(u,j) + \Sigma f(v)$ et la recherche du $\min\{f_j \mid 1 \leq j \leq J(u)\}$ étant les plus coûteuses), et estimer la complexité de l'algorithme en nombre total de développements et d'actualisations de sommets.

Dans ce modèle et pour un hyperlatticiel H fini de N sommets, une analyse de la complexité maximale de AO^* serait :

- tous les sommets de H sont développés par l'algorithme, chacun une seule fois ;

- à la suite du développement d'un sommet u, tous les sommets précédemment développés sont actualisés (en pire cas, tous figurent parmi les ascendants de u) ;

- l'algorithme n'actualise un même sommet qu'au plus une fois dans une itération (contrairement à l'algorithme de Martèlli qui ne profitant pas de l'absence de circuit dans H et n'effectuant pas une actualisation ascendante, peut repasser à plusieurs reprises par le même sommet, ne reportant à chaque fois que l'incrément de coût d'un de ses descendants). Le nombre maximal de développements et d'actualisations est alors:

$$\sum_{k=0}^{N-1} (1+k) = \sum_{1}^{N} k. \qquad : AO^* \text{ est donc en pire cas en } O(N^2).$$

Ce résultat demeure valide pour un hypergraphe infini contenant une solution Λ, mais N désignera alors le nombre de sommets du sous-hypergraphe fini à l'exploration duquel est limité AO^* (i.e., sous-hypergraphe pour lequel $f(u_0) \leq \Psi(\Lambda)$).

La même analyse s'applique à AO_ε. Pour H fini, cet algorighme sera du même ordre de complexité que AO^*. Par contre, dans un hypergraphe infini, AO_ε est moins limité que AO^*, puisqu'il peut poursuivre l'exploration d'un même latticiel tant que $F = f(u_0) \leq (1+\varepsilon)\Psi(\Lambda)$.

Il est donc, en pire cas, plus complexe que AO^* et sa complexité maximale augmente avec ε .

On ne connaît pas de résultat théorique établissant, comme pour A^*, que la complexité maximale d'AO^* est significative de sa complexité moyenne. Par contre, on présente plus loin un modèle empirique du comportement moyen de AO_ε dans un cas particulier. Ce modèle confirme que l'analyse, en pire cas, est non significative de la complexité moyenne de l'algorithme, laquelle diminue avec ε jusqu'à atteindre un seuil ou AO_ε ne développe que les sommets sur le sous-latticiel solution fournie en sortie et ne procède à aucune actualisation (solution sans "backtracking").

Remarque : Si au lieu du critère $\Psi(\Lambda) = \sum m(v).k(v,j)$, on cherche plutôt un SL optimal relativement au critère : $\Phi(\Lambda) = \sum k(v,j)$, on obtient un problème combinatoire beaucoup plus difficile : le problème d'existence correspondant est NP-complet (Sahni [214] , par réduction au problème de la réalisation d'une expression logique).

Notons, cependant, que dans le cas particulier d'un espace de recherche dont tout SL est un arbre, les deux critères Φ et Ψ sont identiques.

°0°

2.6. RECHERCHE HEURISTIQUE & OPTIMISATION COMBINATOIRE

Pour de nombreux problèmes d'optimisation combinatoire et, en particulier, pour tous les problèmes NP-durs, on ne connaît pas d'autre méthode de résolution qu'une énumération implicite dans l'espace des solutions. Une approche générale repose sur le principe d'exploration par séparation et évaluation ("Branch-and-Bound").

Une littérature très abondante a été consacrée aux algorithmes mettant en oeuvre ce principe, parmi laquelle de nombreux articles de synthèse (Cf. par exemple Lawler-Wood (137) , Mitten (161) , Roy (209) , Kohler-Steiglitz (125) , Ibaraki (99) ...). On en reprendra brièvement une formulation simplifiée.

On considère le problème π_o pour lequel on cherche une solution x minimal pour un critère de coût Ψ . On dispose sur ce problème de :

- un moyen de décomposition (on dit aussi séparation) représenté par une arborescence finie B de racine π_o et dont chaque noeud est un sous-problème π_i. Les successeurs de π_i dans B sont les sous-problèmes π_{i_1} , ... , π_{i_k} en lesquels π_i se décompose. On admet que 1) l'ensemble des solutions du sous-problème π_i est partitionné par ceux de ses successeurs, et 2) un noeud π_i sans successeur dans B correspondant à un sous-problème trivial à résoudre (en général, admettant une solution unique) ;

- une fonction d'évaluation f permettant d'estimer le coût de la solution optimale d'un sous-problème. f est supposée être

monotone : $f(\pi_i) \leq f(\pi_j)$ si π_j est un descendant de π_i dans B, et

coïncidente : $f(\pi_i) = \min\{\Psi(x) \mid x \text{ solution de } \pi_i\}$ si π_i est une feuille de B.

Le problème initial π_0 est résolu en construisant progressivement l'arborescence B. L'ensemble des sous-problèmes de B déjà construits, mais non encore décomposés est désigné par $\mathcal{A}$, et s ($\mathcal{A}$) est la fonction d'exploration (caractérisant l'algorithme) qui détermine le sous-problème de $\mathcal{A}$ qui sera décomposé à chaque itération. L'algorithme est le suivant :

Algorithme B&B

Données d'entrée : π_0, moyen de décomposition, fonction d'évaluation f

1. Initialisation : $\mathcal{A} \leftarrow \{\pi_0\}$; $z \leftarrow \infty$
2. Itérer tant que $\mathcal{A} = \emptyset$
2.1. $\pi_i \leftarrow s(\mathcal{A})$
2.2. Si π_i est un sous-problème trivial (ne se décompose pas) alors faire :
résoudre π_i : chercher sa solution optimale x_i
Si $f(\pi_i) < Z$, alors faire : $Z \leftarrow f(\pi_i)$; $x \leftarrow x_i$
2.3. Sinon faire :
Si $f(\pi_i) < z$, alors faire :
Décomposer π_i en générant ses successeurs $\pi_{i_1}, \ldots \pi_{i_k}$
$\mathcal{A} \leftarrow \mathcal{A} \cup \{\pi_{i_1}, \ldots, \pi_{i_k}\} - \{\pi_i\}$
Sinon (éliminer π_i) : $\mathcal{A} \leftarrow \mathcal{A} \{\pi_i\}$
2.4. Fin itération 2
3. Fin.

Données de sortie : x solution optimale de π_0, et $z = \psi(x)$.

Suivant le choix de la fonction d'exploration s, l'algorithme B&B procèdera à un développement en largeur de l'arborescence B, on a un développement en profondeur (PSES, "Backtrack Search"), ou a un développement guidé par la fonction d'évaluation f (PSEP, "Best Search").

On montre (dans plusieurs des références citées) qu'une fonction de sélection de ce dernier type (i.e., $s(\mathcal{A})$ = sous-problème de $\mathcal{A}$ à f minimale) est optimale au sens du nombre-total d'itérations de l'algorithme. On se restreint, par la suite, uniquement à ce cas. La condition d'arrêt de l'algorithme peut alors se simplifier (il y a arrêt dès qu'un sous-problème trivial est résolu en (4.2.2.) et les tests $f(\pi_i) < z$ peuvent être supprimés).

Si l'on transpose la terminologie de B&B à celle d'A* (sous-problème - état, décomposition - développement, sous-problème trivial - état terminal, ...), la similitude entre les deux algorithmes apparaît nettement. Les principales différences semblent être les suivantes :

- B&B explore une <u>arborescence</u>, alors qu'A* s'applique à un <u>graphe</u> quelconque ;

- B&B nécessite une évaluation <u>monotone</u> et coïncidante, alors qu'A* se contente d'une heuristique simplement <u>minorante</u> ;

- A* ne considère que des critères <u>additifs</u> sur des coûts <u>positifs</u> ou nuls (et il décompose la fonction d'estimation du coût d'une solution optimale correspondante à un état u en $f(u) = g(u) + h(u)$), alors que dans B&B, le critère et la structure du coût peuvent être à priori quelconques.

Cette dernière différence n'est, en fait, qu'apparente.

En effet, considérons un graphe G dont chaque arc est valué par un coût K(u,v) positif ou nul, et chaque sommet <u>terminal</u>, $w \in W$, par un coût K(w) <u>quelconque</u>. On cherche dans G un chemin de u_o à un sommet terminal, $(u_o, u_1, \ldots, u_k, w)$, minimisant le critère additif : $K(u_o,u_1) + K(u_1,u_2) + \ldots + K(u_k,w) + K(w)$.

Une heuristique h est dite minorante si $h(u) \leq h^*(u) = \min \left\{ K^*(u_o,w) + K(w) \mid w \in W \right\}$ et coïncidente si $\forall w \in W : h(w) = K(w)$.

On suppose que si G est infini, alors :

1) les coûts de ses sommets terminaux sont bornés inférieurement,

2) G ne contient pas de chemin infini et de coût fini, et

3) il contient un chemin fini de u_o à un sommet terminal. On peut alors énoncer :

Proposition 2.22. : L'algorithme A^* avec une heuristique h minorante et coïncidante est admissible pour le problème de recherche d'un chemin dans le graphe G précédent.

Preuve : On reprend celles des propositions (2.2), (2.3) et (2.4). Il suffit de modifier la définition du sous-graphe fini G' : engendré par le sous-ensemble de sommets $\{ u \in U \mid f^*(u) \leq f^*(u_o) \}$. Tout sommet ne figurant pas dans G' ne peut être développé. D'où l'arrêt de l'algorighme sur un sommet terminal $\hat{u}$ tel que $f(\hat{u}) = \min\{f(u) \mid u \in P\} \leq f^*(u_o)$ or, par définition, $f^*(u_o) = \min\{ f^*(w) \mid w \text{ terminal}\} \leq f^*(\hat{u}) \leq f(\hat{u})$. □

Proposition 2.23. : L'algorithme B&B explorant une arborescence B finie, avec une fonction d'évaluation f, est identique à l'algorithme A^* explorant la même arborescence avec l'heuristique $h(\pi_i) = f(\pi_i)$ et la distribution de coût $K(\pi_i, \pi_j) = 0$ sur tout arc, et $K(\pi_s) = f(\pi_s)$ sur tout sommet terminal π_s.

Preuve : On remarque que :

1) La distribution de coût $K(\pi_i, \pi_j) = 0$ pour tout arc implique que $g(\pi_i) = 0$ et seuls les sommets terminaux ont un coût $f(\pi_i) = h(\pi_i)$;

2) h = f étant monotone, A^* développera à chaque itération le sommet $\hat{u} = s(\mathcal{A})$; et

3) L'absence de circuit dans l'arborescence assure qu'un sommet développé par A^* ne sera pas remis pendant : l'ensemble Q est inutile, et P (dans A^*) joue le même rôle que $\mathcal{A}$ dans B&B.

Il ne reste plus, pour établir l'équivalence des deux algorithmes, qu'à identifier leurs étapes respectives deux à deux. □

On avait vu que, dans le cas particulier où l'heuristique h est monotone, A^* ne développe un même sommet qu'au plus une fois (proposition 2.8.), et donc explore un graphe G selon une arborescence. Comment se compare, dans ce cas, cette exploration avec celle de B&B ? Illustrons le comportement des deux algorithmes sur un exemple.

Exemple 2.7. : Reprenons le graphe G de l'exemple 2.1. avec les coûts et l'heuristique indiqués figure 2.1., et admettons pour B&B l'évaluation f = g + h. La figure 2.5. présente l'arborescence développée par B&B, les sommets développés également par A^* y sont indiqués en traits renforcés.

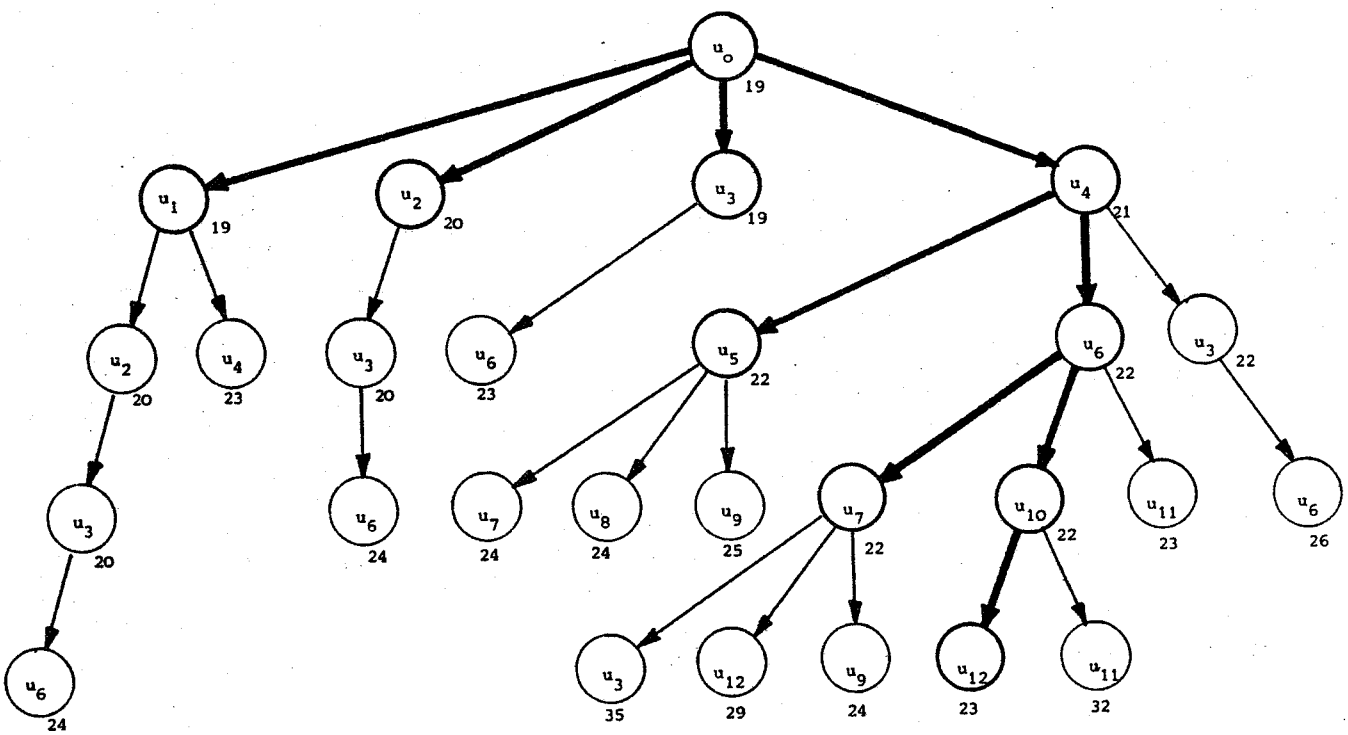


FIGURE 2.5.

On remarque, sur cet exemple, que : 1) les deux algorithmes aboutissent à la même solution, 2) A^* développe un sous-arbre de la structure explorée par B&B (il est donc moins complexe), et 3) B&B peut développer inutilement le même sommet plusieurs fois (cas de u_2 et u_3).

Ibaraki (101)présente une généralisation des algorithmes de "Branch-and-Bound" sur la base d'une représentation dans l'espace d'état d'un problème générique d'optimisation dans le monoïde libre. Cette généralisation porte principalement sur l'introduction d'une relation d'équivalence entre sous-problèmes, qui permet à l'algorithme de reconnaître qu'un sous-problème π_j équivalent à celui en train d'être décomposé a déjà été exploré (le test 2.3.1. devient : (2.3.1.)' : Si il n'existe aucun sous-problème π_j déjà décomposé tel que π_i équivalent à π_j , alors faire : ...). Ibaraki continue, néanmoins, de supposer que la fonction d'évaluation est monotone, ce qui garantit que l'évaluation d'un problème déjà décomposé ne peut être améliorée par la suite, et donc restreint la recherche à une arborescence. On peut alors établir la proposition :

Proposition 2.23. : A^* sur un graphe G muni d'une heuristique h monotone est identique à B&B généralisé avec relation d'équivalence, sur le même graphe et avec l'évaluation f = g + h.

Preuve : Lorsque h est monotone et coïncidente, alors pour tous les sommets développés par A^*, f est aussi monotone et coïncidente (Proposition 2.10). Dans ce cas, le test (2.3.1.)' garantit que B&B ne développera un même sommet qu'au plus une fois et suivra la même arborescence qu'A^*. □

L'algorithme B&B ne se distingue, en définitive d'A^*, que par le mécanisme de remise d'un état déjà développé dans la liste des pendants lorsque l'évaluation de cet état s'est améliorée (ce qui ne peut se produire sur un arbre ou dans le cas d'une heuristique monotone). B&B apparaît donc comme un cas particulier d'A^*.

Il est intéressant de situer la recherche heuristique par rapport à la deuxième grande approche de résolution par énumération implicite basée sur le principe d'optimalité de Bellman et la programmation dynamique.

Sur ce point, plusieurs études dues à Martelli et Montanari (145,148) ont été publiées. Selon ces auteurs, un algorithme de "Branch-and-Bound" n'est pas en mesure de reconnaître qu'il réexplore plusieurs fois le même état (ce qui n'est pas le cas dans la généralisation d'Ibaraki(101)), alors que la programmation dynamique a l'avantage de n'explorer un même état qu'une seule fois. Malheureusement, elle explore systématiquement tous les états, et se trouve incapable de mettre à profit une information heuristique pour restreindre la recherche. Martelli et Montanari (145) montrent qu'on peut y remédier, dans certains cas, en ramenant un problème formulé en terme de programmation dynamique à un problème de recherche de chemin minimal dans un graphe (ou d'un sous-arbre dans un hypergraphe), et d'utiliser pour le résoudre un algorithme tel que A^* ou AO^*. Soulignons, cependant, que le graphe auquel ils se ramènent est de dimension exponentielle en celle des données d'entrée.

Concluons ce chapitre en notant que l'approche d' ε-optimisation est connue depuis longtemps dans les algorithmes de "Branch-and-Bound", et qu'elle s'est révélée souvent fructueuse (126). Il semble curieux que l'extension du Branch-and-Bound à une recherche non arborescente et pour une heuristique non monotone n'ait pas été rapidement accompagnée de la généralisation équivalente de A^* ou AO^* à des algorithmes ε- admissibles. A ma connaissance, aucun travail de ce type n'a été publié, à l'exception des résultats récents dus à Pearl (177,179-182) . Outre diverses analyses d'A^* (celles déjà citées (98), ainsi qu'une étude du comportement d'A^* en fonction de distributions de probabilité de l'écart $h-h^*$), Pearl s'est intéressé à la recherche ε - admissible pour laquelle il a proposé un algorithme nommé A^{**}_{ε} . A chaque itération, cet algorithme développe, parmi l'ensemble des sommets pendants u tel que $f(u) \leq f(\hat{u})$, celui à h minimal. Il s'agit là, en fait, de l'application d'une stratégie opportuniste et en utilisant un seuil d'acceptabilité qui ne correspond pas à la valeur maximale du minorant connue. Le comportement de A^{**}_{ε} a été étudié empiriquement sur le problème du voyageur de commerce (181) .

2.7. CONCLUSION

Ce chapitre a été consacré à des procédures générales de recherche heuristique. On y a repris, avec des améliorations et des corrections, la présentation et l'analyse des algorithmes admissibles A^* et AO^* ; et on y a développé deux algorithmes originaux ϵ-admissibles : A_ϵ et AO_ϵ .

L'avantage de l'approche ϵ-admissible, qui n'apparaît pas sur les résultats de complexité maximale, sera illustré au Chapitre 4 par la présentation d'un modèle empirique du comportement moyen d'AO_ϵ dans le cas particulier de l'optimisation des PDF. Le chapitre suivant développera cette optimisation.

°0°

CHAPITRE 3

SCHEMA D'APPROXIMATION POUR PROCESSUS DECISIONNELS FERMES

3.1. INTRODUCTION

3.2. UN SCHEMA D'APPROXIMATION POUR PROCESSUS DECISIONNELS FERMES

3.2.1. Un espace de recherche

3.2.2. Un estimateur heuristique

3.2.3. Un schéma d'approximation

3.2.4. Post-optimisation : améliorations d'un arbre de décision

3.3. GENERALISATIONS PARAMETRIQUES

3.3.1. Modèles des coûts conditionnels et des coûts dépendants

3.3.2. Modèles des coûts variables

3.3.3. Modèles des coûts d'accès

3.3.4. Modèles des coûts de maintenance

3.4. GENERALISATIONS STRUCTURELLES

3.4.1. Restriction du domaine des caractéristiques et contraintes de succession

3.4.2. Règle Autre

3.4.3. Systèmes de règles récursifs interconnectés

3.5. CONCLUSION

°0°

3.1. INTRODUCTION

Ce chapitre présente un schéma d'approximation pour processus décisionnels fermés comme cas particulier de l'algorithme AO_ε. On introduit,tout d'abord, un hyperlatticiel H correspondant à l'espace de recherche de notre problème et on définit une distribution de coût sur H et une fonction heuristique h pour le modèle des coûts constants.

Les propriétés de monotonie et de coïncidence de h sont établies.

Profitant de la structure particulière du problème, on développe diverses simplifications de l'algorithme AO_ε et on discute plusieurs stratégies d'exploration : heuristique h_c, problème de backtracking, stratégie persévérante, amélioration de ε. Finalement, on propose un algorithme d'amélioration d'un arbre de décision par des techniques de post-optimisation.

Le reste du chapitre est consacré à l'analyse détaillée de diverses généralisations. On passe en revue les six modèles de coût de PDF introduits en (1.4.), proposant et discutant pour chacun une heuristique pour guider la recherche et quelques modifications des algorithmes de base. L'exemple du modèle des coûts d'accès est particulièrement intéressant : on utilise un premier schéma d'approximation sur un problème NP-Complet (dual du "sac-à-dos") pour obtenir une heuristique α-minorante, laquelle guidera la recherche ϵ-admissible d'un arbre de décision.

Trois généralisations au modèle initial des PDF sont ensuite étudiées : systèmes avec contraintes de successions, avec règle "Autre", ou plusieurs systèmes récursifs interconnectés.

3.2. SCHEMA D'APPROXIMATION POUR PROCESSUS DECISIONNELS FERMES

3.2.1. Un espace de recherche

On a présenté au chapitre 1 divers algorithmes (A1 - A4) pour la génération d'arbres de décision représentant un système de règles S. A l'exception de A2 (qui résoud une équation récurrente par programmation dynamique), tous ces algorithmes procèdent par exploration de l'arborescence $\mathcal{T}$, i.e., de l'ensemble des sous-arbres partiels représentants S. Ils effectuent tous un même traitement de base qui consiste à déterminer le r-cube u correspondant à une branche pendante d'un arbre partiel τ, et suivant que $u \in D$, $|R/u| = 1$, ou $|R/u| > 1$, à élaguer cette branche, à la prolonger par une feuille, ou bien à la prolonger par un noeud contenant une caractéristique de X/u. L'itération principale de A1 ou de A4, ou la récursion de A3 sont donc complètement spécifiées par la donnée de u. Or, aux branches pendantes de deux arbres partiels distincts de $\mathcal{T}$, peut correspondre le même r-cube u. Ainsi, par exemple, le (n-2)-cube $u = (0, 1, \varphi, \ldots, \varphi)$ correspond aux branches pendantes des deux sous-arbres suivants :

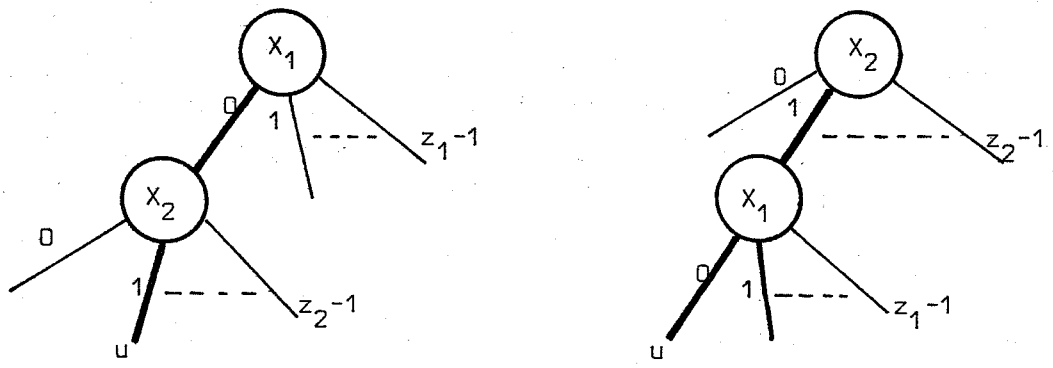


FIGURE 3.1.

Généralement, à un chemin dans un arbre (de la racine à une branche pendante) correspond un terme u unique, mais à u peuvent correspondre des chemins distincts dans (n-k) ! arbres différents. Ceci est dû au fait que u ne reflète pas l'ordre dans lequel les (n-k) caractéristiques spécifiées de u ont été testées.

Pour éviter d'avoir à refaire dans un algorithme d'optimisation plusieurs fois les mêmes itérations, il faut donc être en mesure de reconnaître qu'un terme u a déjà été exploré. Ceci est réalisé au niveau de l'implémentation de A 3 ou de A 4 par l'introduction des relations d'équivalences du type de celle d'Ibaraki (99) . Martelli et Montanari (149) proposent de porter cette reconnaissance dans l'espace de recherche lui-même, et de contracter l'arborescence à un graphe ET/OU, graphe qu'ils explorent alors par leur algorithme admissible HS (du type d'AO*).

Dans la terminologie du chapitre précédent, cet espace de recherche est un hyperlatticiel $H = (U, \mathcal{E})$ dont :

- la racine est le n-cube $u_o = \left(\sum_{i=1}^{n} z_i\right)$;

- chaque terme u de U est un sommet de H ;

- du sommet u part l'ensemble de connecteurs $\{\Gamma_i(u) \mid i \in \Phi(u)\}$;
  le connecteur $\Gamma_i(u)$ reliant u à ses z_i sous-termes (u/i ← j) obtenus en fixant la première coordonnée de u à j ; j = 0, 1, ..., z_i-1.
  D'après cette définition, il ne part aucun connecteur des 0-cubes, lesquels sont pris pour sommets terminaux.

Exemple 3.1. : On présente, en figure 3.2., une partie de l'hyperlatticiel H correspondant à un système de quatre caractéristiques binaires, en précisant les sommets issus de (1 φ φ φ), (φ 0 φ φ) et (φ φ 0 φ).

Il n'est pas difficile de vérifier que H ainsi défini est bien un hyperlatticiel. En fait, le graphe associé à H est le latticiel Dedekindien du n-cube $\left(\sum_{i=1}^{n} z_i\right)$. Il lui correspond l'ordre partiel des rangs sur U suivant : la racine u_o est de rang 0, le r-cube u est un sommet de rang (n-r), et tous ses successeurs sont au rang (n-r+1).

Pour refléter les particularités d'un système de règles $S = \{X, A, D, R\}$, on précise légèrement la définition de H comme suit :

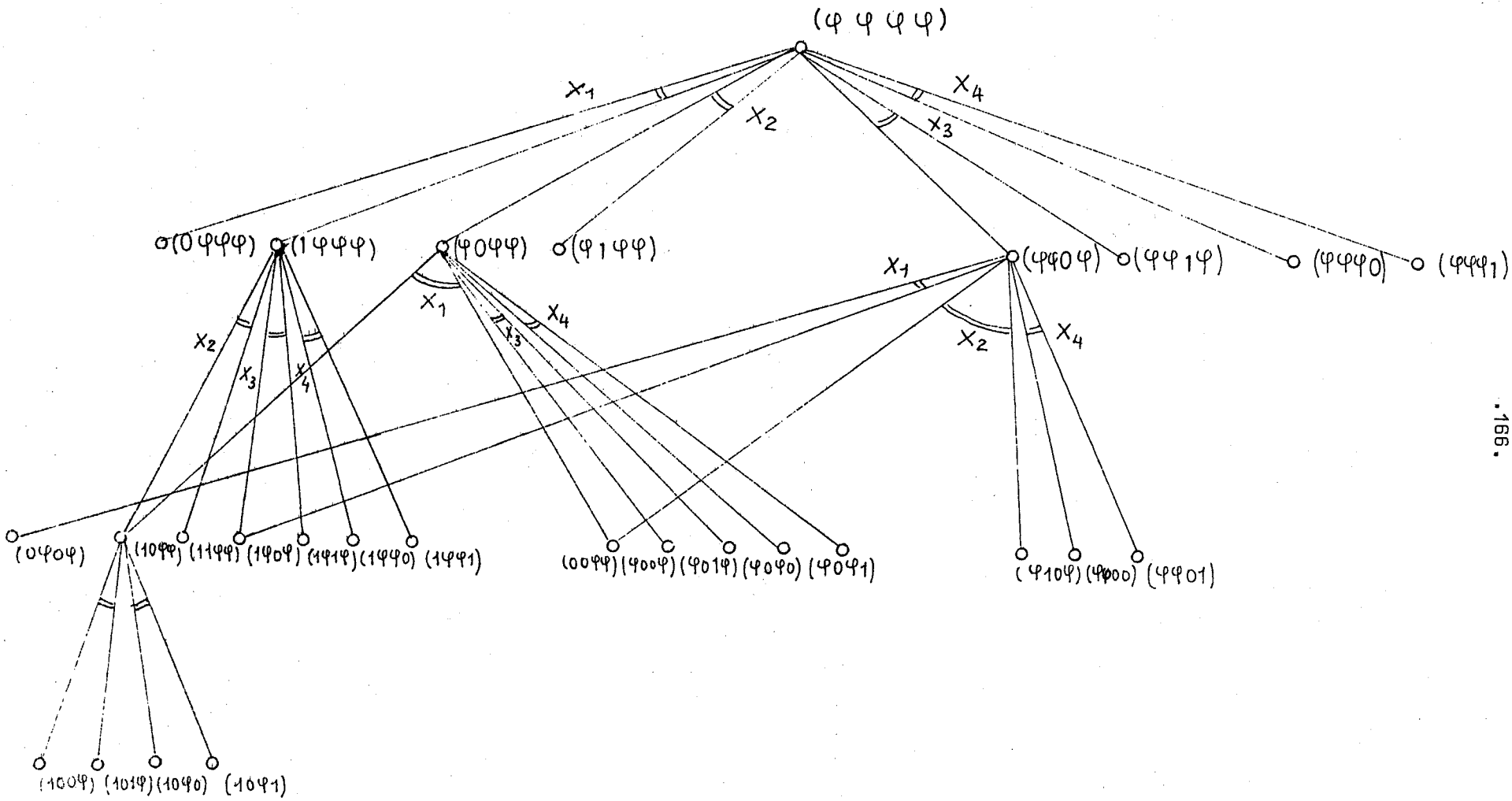

(φ φ φ φ)
X1
X2
X3
X4
σ(0φφφ)
(1φφφ)
(φ0φφ)
(φ1φφ)
(φφ0φ)
(φφ1φ)
(φφφ0)
(φφφ1)
(0φ0φ)
(10φφ)
(11φφ)
(1φ0φ)
(1φ1φ)
(1φφ0)
(1φφ1)
(00φφ)
(φ00φ)
(φ01φ)
(φ0φ0)
(φ0φ1)
(φ10φ)
(φφ00)
(φφ01)
(100φ)
(101φ)
(10φ0)
(10φ1)

FIGURE 3.2.

- l'ensemble des sommets terminaux est :

$W = \{ u \in U \mid u \in D \text{ ou } |R/u| = 1 \}$. Cet ensemble inclut bien entendu tous les 0- cubes.

- l'ensemble des connecteurs issus d'un terme u non terminal est :

$$\{ \Gamma_i(u) \mid X_i \in X/u \} =$$

$$\{ \Gamma_i(u) \mid i \in \Phi(u) \text{ et } \exists j, k ; j \neq k \text{ tel que : } (u/i \leftarrow j) \notin D \text{ et } (u/i \leftarrow k) \notin D \}.$$

Cette restriction permet de supprimer les connecteurs n'ayant pas plus de un sommet successeur autre que ceux de D (ce qui est équivalent à supprimer dans un arbre les noeuds à une seule branche).

On définit également la structure de coût de H dans le modèle des coûts constants, et relativement aux distributions ρ et μ de S par :

$$K(u, i) = \rho_i \cdot \mu(u)$$

On peut alors énoncer la :

Proposition 3.1. : Tout sous-latticiel Λ de H est un arbre t représentant S, et son coût est: $\Psi(\Lambda) = \Psi(t)$.

Preuve : Par définition, pour tout sommet u d'un SL Λ , ou bien u est terminal, ou bien de u est issu un connecteur unique dans Λ. Etiquetons un sommet u de Λ dont est issu le connecteur $\Gamma_i(u)$ par la caractéristique X_i (ce connecteur correspond bien à une caractéristique de X/u).

A un sommet terminal u tel que $|R/u| = 1$, associons l'action A_j correspondant à la règle unique de R/u ; et si $u \in D$, élaguons ce sommet terminal, ainsi que toutes les arêtes de Λ qui y parviennent. On peut alors établir l'équivalence des algorithmes A1 et SOUS-LATTICIEL. S'ils effectuent tous deux le même choix non déterministe (lorsque SOUS-LATTICIEL choisit pour u le connecteur $\Gamma_i(u)$, A1 choisit pour le même terme la caractéristique X_i dans X/u), alors une itération de SOUS-LATTICIEL est identique à une itération de A1, à la différence près que A1

réserve une itération propre aux sommets terminaux. Il s'ensuit que le SL défini par SOUS-LATTICIEL est identique à l'arbre t défini par A1, arbre qui représente bien S (proposition 1.5.).

Finalement, l'expression (1.4.3.-ii) du coût d'un arbre s'identifie avec la définition du coût d'un SL. □

L'espace de recherche H comporte au maximum $|U| = \prod_{i=1}^{n} (1 + z_i)$ sommets. Pour déterminer le nombre maximum de connecteurs $|\mathcal{E}|$ de H, il suffit de revenir à l'analyse de l'algorithme A2 en remarquant que la programmation dynamique parcourt, d'une manière ascendante, le graphe associé à H, et qu'il y a autant de connecteurs issus d'un sommet u de H que de comparaisons dans la minimisation de l'équation récurrente (1.1.2.) de A2. L'expression (1.6.1.-iv) donne alors le nombre total de comparaisons :

$$|\mathcal{E}| = \sum_{k=1}^{n} \prod_{\substack{j=1 \\ j \neq k}}^{n} (1 + z_i)$$

Il est intéressant, pour estimer la contraction de l'espace de recherche obtenue, de comparer $|U|$ ou $|\mathcal{E}|$ à $|\mathcal{T}|$ (1.5.1.-i). A titre d'exemple, pour 12 caractéristiques binaires, on passe d'un espace de recherche arborescent $\mathcal{T}$ de 3.36×10^{902} noeuds à un hyperlatticiel H de 5.3×10^{5} sommets. Ceci provient du fait que tout arbre (partiel ou complet) est représenté comme un noeud explicite de $\mathcal{T}$, alors qu'il ne figure que comme un sous-arbre de H.

3.2.2. Un estimateur heuristique

Pour permettre une exploration efficace de H par un schéma d'approximation du type d'AO$_\varepsilon$, on a besoin d'une fonction d'estimation heuristique minorante sur U. On définira cette fonction à partir d'une procédure qui permettra de déterminer pour une caractéristique X_i, quelle est la probabilité d'avoir à ne pas évaluer X_i dans le processus décisionnel.

Rappelons nos hypothèses :

- $S = \{X, A, D, R\}$ est un système de règles consistant et complet, et dont les termes d'une même règle sont <u>sans recouvrement</u> ; les seuls

recouvrements permis sont entre termes appartenant à des règles distinctes et dont l'intersection est dans D, ou entre termes de D (le non recouvrement des termes de D n'est nécessaire qu'à la vérification en temps polynômial de la complétude de S) ;

- On dispose d'une distribution de probabilité μ sur U.

La procédure RECOUVREMENT définit pour chaque caractéristique X_i un sous-ensemble C_i de r-cubes où X_i est indifférente :

Algorithme RECOUVREMENT

Données d'entrée : $S = \{X, A, D, R\}$

1. Itérer sur $\{X_i \in X\}$

1.1. $C_i \leftarrow \emptyset$

1.2. Itérer sur $\{R_j \in R\}$

1.2.1. $C_i \leftarrow C_i \cup \{u \in R_j \mid i \in \Phi(u)\}$

1.2.2. $P \leftarrow \{u \in (R_j \cup D) \mid i \notin \Phi(u)\}$

1.2.3. Partitionner P en z_i sous-ensemble $P_0, \ldots, P_l, \ldots, P_{z_i-1}$, avec
$P_l \leftarrow \{u \in P \mid u(i) = j\}$

1.2.4. Itérer sur $\{(v_0, \ldots, v_l, \ldots, v_{z_i-1}) \in P_0 x \ldots x P_l x \ldots x P_{z_i-1}\}$
$u \leftarrow \bigcap_{0 \leq l \leq z_i} (v_l / i \leftarrow \varphi)$
Si $u \neq \emptyset$, alors $C_i \leftarrow C_i \cup \{u\}$
Fin itération 1.2.4.

1.2.5. Fin itération 1.2.

1.3. Fin itération 1

Donnée de sortie : $C_1, \ldots, C_i, \ldots, C_n$.

Proposition 3.2. : $q_i = \sum_{u \in C_i} \mu(u)$ est la probabilité de se trouver dans un état $e \in E$ pour lequel il est possible de déterminer la règle de décision valide sans avoir à évaluer la caractéristique X_i ; et $q_i(u) = \sum_{v \in C_i} \mu(v \cap u)$ est la probabilité jointe d'avoir à la fois l'événement précédent et $(X_1(e), \ldots, X_n(e))cu$.

Preuve : Le processus décisionnel n'évaluera pas X_i s'il traverse t, un arbre représentant S, le long d'un chemin où ne figure pas X_i, i.e., un chemin représenté par un terme u ayant $u(i) = \varphi$. Or, par définition, t représente S ssi t réalise une partition de $(\sum_{i=1}^{n} z_i)$ en (a+1) blocs inclus respectivement dans $(R_1 \cup D), \ldots, (R_a \cup D)$, D ; le dernier bloc pouvant être vide.

L'algorithme RECOUVREMENT détermine pour chacun des a 1er blocs tous les termes distincts pouvant avoir $u(i) = \varphi$. Il s'agit de ceux ayant cette propriété dans la représentation initiale de S (1.3.1.), et de ceux qui peuvent être obtenus par regroupements de z_i termes adjacents le long de leur $i^{ème}$ coordonnée, chacun étant un sous-terme d'un élément de bloc $R_j \cup D$. La preuve se complète en remarquant que du fait de nos hypothèses sur S, $\forall u, v \in S$, ou bien $u \cap v = \emptyset$, ou bien $u \cap v \subset D$, et dans les deux cas $\mu(u \cap v) = 0$. □

Analyse de RECOUVREMENT : Si $m = \max\{|R_j \cup D| \mid 1 \leq j \leq a\}$, la partition de P se fait en m opérations, et l'itération (1.2.4.), sur tous les z_i-tables des blocs de cette partition, se fera au plus $(m/z_i)^{z_i}$ fois. L'intersection de z_i termes nécessite $(n \times z_i)$ opérations. Compte-tenu des itérations sur les a règles et les n conditions, l'algorithme exécutera au plus $[\sum_{i=1}^{n} a \times n \times z_i \times (m/z_i)^{z_i}]$ instructions élémentaires. Sa complexité sera donc en $O(a \times n^2 \times m^z)$; $z = \max\{z_i\}$.

Remarque : Le problème abordé par RECOUVREMENT est similaire à celui de la minimisation des fonctions booléennes (lequel est NP-Complet (214)). Il s'en distingue, néanmoins, par le fait qu'on n'a pas besoin de la représentation minimale de la fonction $(R_j \cup D)$, mais uniquement de ses monômes admettant X_i comme variable consensus.

On peut définir, à présent, une fonction d'évaluation heuristique sur l'ensemble U des sommets de H :

$$u \in U \rightarrow h(u) = \sum_{i \in \Phi(u)} p_i \left(\mu(u) - q_i(u)\right)$$

Proposition 3.3. : h (u) est une fonction d'évaluation, monotone et coïncidente du coût d'un SL $\Lambda(u)$ issu d'un sommet u, i.e. du coût d'un sous-arbre de décision représentant S/u.

Preuve :

1) Monotonie de h : on veut montrer que $\forall j \in \Phi(u)$, on a :

$$h(u) \leq K(u,j) + \sum_{v \in \Gamma_j(u)} h(v).$$

Les z_j termes $v \in \Gamma_j(u)$ constituent une partition de u (au sens des ensembles d'états du système appartenant à chacun d'entre eux), on a donc :

$\mu(u) = \sum_{v \in \Gamma_j(u)} \mu(v)$, et de même pour les probabilités jointes :

$q_i(u) = \sum_{v \in \Gamma_j(u)} q_i(v)$, si $i \neq j$.

D'où : $$\sum_{v \in \Gamma_j(u)} h(v) = \sum_{v \in \Gamma_j(u)} \sum_{i \in \Phi(v)} \rho_i \left(\mu(v) - q_i(v)\right)$$

Or, $\forall v \in \Gamma_j(u)$: $\Phi(v) = \Phi(u) - j$. On a donc :

$$\sum_{v \in \Gamma_j(u)} h(v) = \sum_{\substack{i \in \Phi(u) \\ i \neq j}} \rho_i \sum_{v \in \Gamma_j(u)} \left(\mu(v) - q_i(v)\right)$$

$$= \sum_{i \in \Phi(u)} \rho_i \left(\mu(u) - q_i(u)\right) \quad . \text{ On en déduit :}$$

$$K(u,j) + \sum_{v \in \Gamma_j(u)} h(v) = h(u) + \rho_j \quad q_j(u)$$

h est donc bien monotone.

2) Coïncidence de h : u est terminal par définition si $u \in D$ ou si $|R/u| = 1$.

- si $u \in D \Rightarrow \mu(u) = 0$ et $q_i(u) = o \Rightarrow h(u) = 0$
- si $|R/u| = 1 \Rightarrow \exists j$ unique tel que $u \subset (R_j \cup D)$. Donc, ou bien $\Phi(u) = \emptyset$ et $h(u) = 0$; ou bien $\forall i \in \Phi(u)$ l'ensemble C_i contient u ou contient une décomposition de u en termes adjacents, et dans les deux cas, $q_i(u) = \mu(u)$ et $h(u) = 0$. □

Remarques :

1) La preuve de la monotonie de h a permis également de montrer que l'incrément d'estimation, après développement d'un sommet u, est :

$$\min_{j \in \Phi(u)} \left\{ K(u,j) + \sum_{v \in \Gamma_j(u)} h(v) \right\} - h(u) = \min_{j \in \Phi(u)} \left\{ \rho_j , q_j(u) \right\}$$

On mettra à profit cette relation pour simplifier AO_ε.

2) Monotonie et coïncidence assurent que h est aussi minorante. Mais, on peut le prouver directement en notant que :

. $[\mu(u) - q_i(u)]$ est la probabilité pour que X_i soit nécessaire dans S/u ;

. $h(u) = \Sigma \rho_i[\mu(u) - q_i(u)]$ est l'espérance de coût nécessaire de S/u.

En partant de ce qui précède, on peut aborder la question de la redondance d'une caractéristique dans un système S (ou dans un sous-système S/u).

Définition : X_i est une caractéristique redondante dans S ssi il est possible de supprimer cette caractéristique de toutes les règles de S tout en gardant un système consistant et complet (définition similaire en remplaçant S par S/u).

La proposition suivante est immédiate :

Proposition 3.4. : X_i est redondante dans S ssi $q_i = 1$ (X_i est redondante dans S/u ssi $q_i(u) = \mu(u)$).

Il est donc possible, si X_i est redondante dans S, de trouver un arbre de décision représentant S et ne contenant X_i dans aucun de ses noeuds. Mais un tel arbre est-il nécessairement meilleur en coût qu'un autre arbre contenant X_i ? Plus précisément, la suppression de X_i de S n'entraîne-t-elle aucune incidence sur le coût de l'arbre optimal représentant S. La réponse est positive, comme le montre le contre-exemple très simple suivant :

Exemple 3.2. : S contient quatre règles et trois caractéristiques binaires :
$R_1 = \{(1,1,0)\}$; $R_2 = \{(1,0,0)\}$; $R_3 = \{(0,0,0)\}$; $R_4 = \{(0,1,1)\}$ et
$D = \{(1,1,1)$; $(1,0,1)$; $(0,0,1)$; $(0,1,0)\}$.

On remarque immédiatement que X_3 est redondante. Néanmoins, pour les distributions de coût $\rho_1 = \rho_2 = \rho_3 = 10$, et de probabilité $\mu(1,1,0) = 0.1$; $\mu(1,0,0) = 0.2$; $\mu(0,0,0) = 0.3$ et $\mu(0,1,1) = 0.4$, l'arbre optimal représentant S sans X_3 est t_1 de coût $\psi(t_1) = 20$, alors qu'avec X_3, on trouve l'arbre t_2 de coût $\psi(t_2) = 19$.

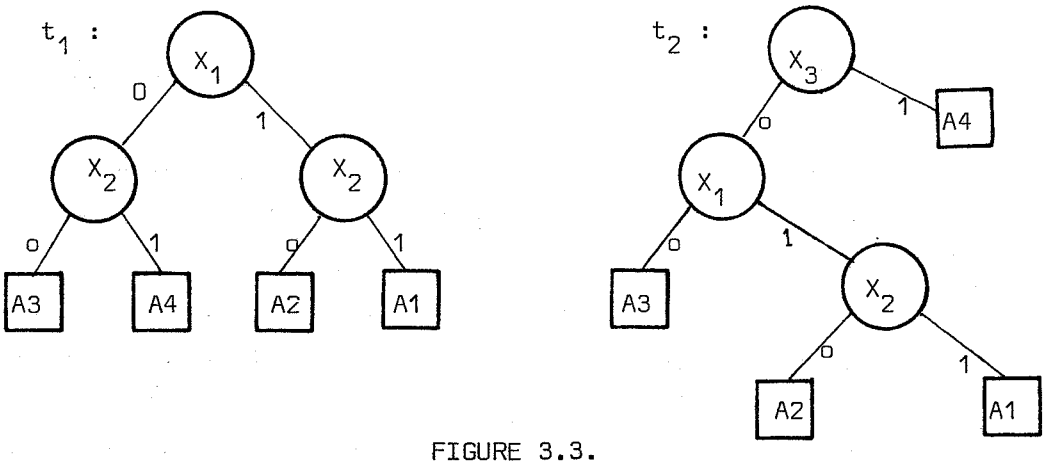


FIGURE 3.3.

L'identification parmi les caractéristiques redondantes dans un systèmes S (ou dans un sous-système S/u), de celles qui pourront à priori être supprimées sans aucun risque de perte d'optimalité, est cependant très importante, car elle permettra de simplifier considérablement la recherche dans H. Aussi, on dira que X_i est une caractéristique complètement redondante dans S ssi X_i est redondante, et pour tout arbre t contenant X_i et représentant S, il existe un autre arbre t' ne contenant pas X_i tel que, quelles que soient les distributions p et μ : $\Psi(t') \leq \Psi(t)$. On peut alors établir la condition nécessaire et suffisante suivante :

Proposition 3.5. : X_i est complètement redondante dans S ssi l'un des deux cas suivants est vrai :

i) les sous-systèmes $S/(X_i = 0)$, ..., $S/(X_i = j)$, ..., $S/(X_i = z_i - 1)$ sont identiques ; ou

ii) pour un seul des sous-systèmes précédents : $R/(X_i = j) \neq \emptyset$.

Preuve (par l'absurde) : Il est clair que si X_i vérifie (i) ou (ii), alors elle est redondante. Soit t un arbre optimal représentant S et contenant X_i. Si X_i vérifie (ii), alors de tout noeud de t contenant X_i ne part qu'une seule branche (celle correspondant à la valeur j). La suppression de ce noeud (et la liaison de son prédécesseur à son successeur), conduit à un arbre t' qui représente toujours S, mais $\Psi(t') \leq \Psi(t)$ et t n'est donc pas optimal.

Si X_i vérifie (i), alors les sous-arbres de t correspondant à chacune des branches du noeud X_i sont identiques, et de nouveau ce noeud et tous ses sous-

arbres, sauf un pouvant être supprimé en obtenant toujours un arbre représentant S et de coût inférieur à celui de t.

Réciproquement, soit X_i une caractéristique complètement redondante, et t un arbre contenant X_i (admettons, pour simplifier, que X_i est sa racine).

Par définition, il existe t' ne contenant pas X_i tel que : $\forall \rho$ et μ : $\Psi(t') \leq \Psi(t)$. $\Psi(t)$ contient donc tous les termes qui interviennent dans le développement formel de $\Psi(t')$: t' est un sous-arbre de t. Or, t' représente également S_i, donc tout noeud de t non compris dans t' correspond à une caractéristique redondante, et on peut passer de t à t' en enlevant un tel noeud. Finalement, les seuls cas où un noeud peut être supprimé sont ceux de la proposition. □

On généralise simplement ce qui précède à la redondance complète dans un sous-système S/u en remarquant, cependant, qu'une caractéristique X_i vérifiant le cas (ii) n'appartient pas à S/u par définition.

Un moyen pratique d'identifier si une caractéristique X_i est complètement redondante dans un système (ou sous-système), est d'analyser les termes qui interviennent dans C_i (ou $C_i \cap \{u\}$).

Il y a redondance complète si $\sum_{u \in C_i} \mu(u) = 1$ et si :

- tous les éléments de C_i ont été obtenus par regroupeemnt de z_i termes dont aucun n'appartient à D (cas (i)) ; ou bien

- tous ont été obtenus par regroupement de z_i termes : $v_o, v_1, \ldots, v_{z_i-1}$; et tous les $v_l \in D$, sauf un seul v_j, la valeur j étant commune à tous les termes de C_i (cas (ii)).

On peut modifier la procédure RECOUVREMENT pour simplifier la reconnaissance des caractéristiques complètement redondantes. Pour chaque X_i, on définira trois ensembles de recouvrements C_i, C'_i, C''_i :

-1) dans C_i, on mettra :

. tous les $u \in (S - D)$ tel que $u(i) = \varphi$ et $\forall l$, $0 \leq l < z_i$ $(u/i \leftarrow l) \notin D$;

. tout u obtenu par regroupement d'un z_i-tuple $(v_0, \ldots, v_{z_i-1})$ tel que $\forall l \quad v_l \notin D$;

-2) dans C'_i, on mettra :

. tout $u \in (S - D)$ tel que $u(i) = \varphi$ et $\exists j \; \forall l$, $l \neq j$: $(u/i \leftarrow j) \notin D$, $(u/i \leftarrow l) \subset D$

. tout u obtenu par regroupement sur $(v_0, \ldots, v_{z_i-1})$ tel que un seul v_j ne figure pas dans D ;

. à chaque élément de C'_i, on associe l'indice j de son sous-terme v_j n'appartenant pas à D ;

-3) dans C''_i, on mettra tous les autres regroupements.

Le test de redondance est alors le suivant :

-1) X_i est redondante dans S/u ssi $\sum_{v \in C_i \cup C'_i \cup C''_i} \mu(v \cap u) = \mu(u)$;

-2) X_i est complètement redondante dans S/u ssi :

(i) . soit $\sum_{v \in C_i} \mu(v \cap u) = \mu(u)$

(ii) . soit $\sum_{v \in C'_i} \mu(v \cap u) = \mu(u)$ et il correspond le même j à tous les v de la somme précédente pour lesquels $v \cap u \neq \emptyset$.

Remarques :

1) Si plusieurs caractéristiques sont redondantes dans S, il n'est en général pas possible de les supprimer toutes : la suppression de l'une d'entre elles peut conduire à un système où les autres ne sont plus redondantes. Cependant, on vérifie facilement que la suppression simultanée de toutes les caractéristiques complètement redondantes continue à fournir un système consistant et complet.

2) La définition des connecteurs issus de u, ou celle de X/u, donnée par $X/u = \{ X_i \in X \mid i \in \Phi(u)$, et $\exists j, k$; $j \neq k$: $(u/i \leftarrow j) \notin D$ et $(u/i \leftarrow k) \subset D \}$ est facilement obtenue par la vérification de la redondance complète : si $i \in \Phi(u)$ et $X_i \notin X/u$, alors X_i est complètement redondante dans S/u. La réciproque est fausse : X/u n'est pas exempte de caractéristiques complètement redondantes (dues au cas (i)), même celle correspondante au $\min \{ \rho_j \; q_j(u) \}$ peut l'être. Le test de la redondance complète sera donc nécessaire à un algorithme de recherche dans H.

3) Le calcul direct de $(1 - q_i)$, probabilité pour que X_i soit nécessaire dans S (ou celui de $\mu(u) - q_i(u)$ pour S/u), est plus complexe que celui de q_i : la recherche des termes et sous-termes n'intervenant dans aucun recouvrement par rapport à X_i peut nécessiter un nombre exponentiel d'étapes.

4) Si t_j est un arbre de décision représentant S, tel que tout chemin de t_j n'évalue la caractéristique X_j qu'en dernier lieu, lorsqu'un test est encore nécessaire ($|R/u| > 1$), et qu'aucune autre caractéristique n'est disponible ($X/U = \{X_j\}$), alors $1 - q_j = \sum_{X_j \in \upsilon(\Gamma^{-1}(\ell))} \mu(\ell)$

3.2.3. Un schéma d'approximation pour PDF

On l'obtient directement à partir de AO_ε appliqué à la recherche dans un espace H dont tout SL est un arbre, et avec une heuristique minorante et coïncidante. Développons, ici, quelques simplifications de cet algorithme et une instanciation de certaines de ses stratégies d'exploration, d'actualisation et de "backtracking" en vue d'exploiter la structure particulière du problème.

L'algorithme AO_ε n'utilise pas l'estimation du coût d'un sommet d'une manière absolue, mais uniquement d'une manière incrémentale. On vérifie qu'à une seule exception près (pour la racine u_o), les fonctions f et h n'y interviennent jamais autrement que dans l'expression $(f(u) - h(u))$, ou $(f'(u) - f(u))$. Or, l'heuristique définie précédemment possède l'intéressante propriété d'avoir un incrément d'estimation particulièrement simple à calculer. Cet incrément, noté Δh, est :

$$\Delta h_j(u) = \left[K(u,j) + \sum_{v \in \Gamma_j(u)} h(v)\right] - h(u) = p_j\, q_j(u)$$

Au lieu d'associer à un sommet u le vecteur $(f_{j1}(u), \ldots, f_{jk}(u))$ d'estimation des coûts des SL de racine u sur ces divers connecteurs, $\{j_1, \ldots j_k\} = \Phi(u)$, associons lui plutôt le vecteur $(g_{j1}(u), \ldots, g_{jk}(u))$, avec $g_j(u) = f_j(u) - h(u)$.

Par définition $f_j(u) = K(u,j) + \sum_{v \in \Gamma_j(u)} f(v)$ et $f(u) = \min_j \{f_j(u)\}$.

Si on pose $g(u) = \min\{g_j(u)\}$, on aura : $f(u) = g(u) + h(u)$; d'où :

$$g_j(u) = K(u,j) + \sum_v f(v) - h(u)$$
$$= K(u,j) + \sum_v (g(v) + h(v)) - h(u)$$
$$= h_j(u) + \sum_v g(v)$$

Donc, finalement : $g(u) = \min_j \left\{ \rho_j \, q_j(u) + \sum_{v \in \Gamma_j(u)} g(v) \right\}$; $g'(u)$

correspondra au deuxième minimum de cette expression, et on a :
$f'(u) - f(u) = g'(u) - g(u)$.

Compte-tenu du fait que pour un sommet non encore développé $f(v) = h(v)$, et donc $g(v) = 0$, le calcul de $g(u)$ auquel on s'est ramené est beaucoup plus simple que celui de $f(u) = \min_j \left\{ K(u,j) + \sum_j \sum_i \rho_i (\mu(u) - q_i(v)) \right\}$.

Pour la mise en oeuvre de cette simplification, on se contentera de remplacer partout dans AOε les expressions $(f'(u) - f(u))$ et $f(u_o)$ par $(g'(u) - g(u))$ et $(g(u_o) + h(u_o))$ respectivement. Les procédures INITIALISER et DEVELOPPER définiront ces quantités et permettront également de supprimer de H tout connecteur correspondant à une caractéristique complètement redondante :

INITIALISER :

1. $P \leftarrow \emptyset$; $Q \leftarrow \{u_o\}$; $A \leftarrow \emptyset$; $\Psi' \leftarrow \infty$; $q \leftarrow 0$
2. Itérer sur $\{ 1 \leq j \leq n \}$
   Si X_j est complètement redondante, alors $g_j(u_o) \leftarrow \infty$
   Sinon faire : $g_j(u_o) \leftarrow \rho_j \, q_j$; $P \leftarrow P \cup \{ v \in \Gamma_j(u_o) \}$
   Fin itération 2.
3. $g(u_o) = \min_j \{ g_j(u_o) \}$; $\hat{j} \leftarrow$ indice du minimum précédent
4. $g'(u_o) = \min_{j \neq \hat{j}} \{ g_j(u_o) \}$
5. $F \leftarrow g(u_o) + h(u_o)$
6. $F' \leftarrow g'(u_o) + h(u_o)$
7. $P' \leftarrow \{ v \in \Gamma_{\hat{j}}(u_o) \mid v \notin W \}$
8. Fin

DEVELOPPER (u)

1. $P \leftarrow P - \{u\}$; $Q \leftarrow Q \cup \{u\}$

2. Itérer sur $\{j \in \Phi(u)\}$

2.1. Si X_j est complètement redondante dans S/u, alors $g_j(u) \leftarrow \infty$

2.2. Sinon faire :

Pour $\{v \in \Gamma_j(u) \mid v \notin P \cup Q\}$ faire : $g(v) \leftarrow 0$; $I(v) \leftarrow \emptyset$; $P \leftarrow P \cup \{v\}$

$g_j(u) \leftarrow p_j\, q_j(u) + \sum_{v \in \Gamma_j(u)} g(v)$

2.3. Fin itération 2.

3. $g(u) \leftarrow \min_j \{g_j(u)\}$; $\hat{j}$ indice du minimum

4. $g'(u) \leftarrow \min_{j \neq \hat{j}} \{g_j(u)\}$

5. $F \leftarrow F + g(u)$

6. $P' \leftarrow (P' - \{u\}) \cup \{v \in \Gamma_{\hat{j}}(u) \mid v \notin W\}$

7. Si $g(u) \neq 0$, alors faire :

$A \leftarrow A \cup \{u\}$

Pour $\{v \in \Gamma^{-1}(u)\}$ faire : $I(v) \leftarrow I(v) \cup \{j \mid u \in \Gamma_j(v)\}$

8. Fin.

Les modifications à apporter à la procédure ACTUALISER seront plus substantielles. L'ensemble A est redéfini en : ensemble des sommets dont il faut actualiser les pères (Cf. instruction 7.1. de DEVELOPPER). Par ailleurs, malgré la monotonie de h, une actualisation complète de tous les sommets de H ayant hérité de nouveaux descendants est plus avantageuse qu'une actualisation restreinte aux seuls sommets sur les connecteurs minimaux $\hat{j}$. Les arguments discutés en section 2.4. sont renforcés ici par le fait que la relation d'ordre sur H simplifie l'actualisation ascendante : si $v \in A$ est de rang maximal dans A, alors $A \cap \hat{\Gamma}(v) = \emptyset$. En contre partie, l'utilisation d'un incrément d'estimation, au lieu d'une estimation absolue, conduira à restreindre l'actualisation en ne reportant sur un sommet u que les g(v) de ses successeurs dont il n'a pas été tenu compte lors du développement de u, i.e., ceux développés après u. Il s'agit des descendants de u mis dans A, car la stratégie d'exploration d'AO_ε assure qu'entre deux actualisations, aucun sommet $w \in \hat{\Gamma}^{-1}(u)$ ne sera développé après u.

D'où, finalement, la procédure :

ACTUALISER

1. $k \leftarrow$ rang maximal des sommets de A

2. Itérer tant que $k \neq 0$

2.1. $B \leftarrow \{$ sommets de A de rang k $\}$

2.2. $E \leftarrow \Gamma^{-1}(B)$

2.3. Itérer sur $\{ u \in E \mid I(u) \neq \emptyset \}$

Pour $\{ j \in I(u)$ faire : $g_j(u) \leftarrow g_j(u) + \sum_{v \in B \cup \Gamma_j(u)} g(v)$

$\hat{j} \leftarrow$ indice du min $\{ g_j(u) \}$

$g'(u) \leftarrow \min_{j \neq \hat{j}} \{ g_j(u) \}$; $I(u) \leftarrow \emptyset$

2.3.4. Si $g(u) \neq g_{\hat{j}}(u)$, alors faire :

$g(u) \leftarrow g_{\hat{j}}(u)$

$A \leftarrow A \cup \{u\}$

Pour $\{ v \in \Gamma^{-1}(u) \}$ faire : $I(v) \leftarrow I(v) \cup \{ j \mid u \in \Gamma_j(v) \}$

Fin itération 2.3.

2.4. $A \leftarrow A - B$

2.5. $k \leftarrow k - 1$

2.6. Fin itération 2.

3. Fin.

L'instanciation de la procédure CHOIX, qui à chaque itération sélectionne parmi l'ensemble P' des sommets à développer sur le SL en cours d'exploration celui qui sera examiné, profitera également de la relation d'ordre sur H. Si un sommet v est de rang plus élevé qu'un sommet u, $\mu(v)$ est en général inférieur à $\mu(u)$ (l'inégalité a toujours lieu si $v \in \Gamma(u)$). Aussi, l'incidence d'un sommet sur le coût global d'un SL diminue généralement avec son rang. Sachant que $q_j(u) \leq \mu(u)$, la même remarque s'applique au niveau des estimations. L'acceptabilité du SL exploré sera moins souvent remise en cause par le développement d'un sommet de rang élevé que par celui d'un sommet de rang faible. Ainsi, une exploration, dans l'ordre de Tarry par exemple, en laissant des sommets de rang très faible non développés, entraînera le plus souvent des développements inutiles. On développera donc les sommets de P' par ordre de rang croissant et pour les sommets de même rang, par

ordre transverse. Remarquons que cette instanciation de CHOIX fait qu'une exploration en "profondeur" de H correspond en fait à un développement en "largeur" d'un arbre .

Par ailleurs, il est facile d'établir que si pour un sommet u $|R/u| \geq 1$ et $|X/u| = 1$, alors $q_j(u) = 0$ pour $\{j\} = \Phi(u)$.

Un tel sommet n'entraîne donc pas d'incrément sur l'estimation F, et tous ses fils (sur son connecteur unique) sont terminaux ; sa procédure de développement peut donc être simplifiée. Notons que cette propriété renforce les considérations précédentes sur l'instanciation de CHOIX.

Une propriété symétrique à la précédente, et également très simple à démontrer, établit que si un sommet u a tous ses fils le long d'un connecteur $\Gamma_j(u)$ homogènes, alors ce connecteur correspond à un incrément d'estimation nul, I.e., si $\exists j \in \Phi(u)$ tel que $\forall v \in \Gamma_j(u) : |R/u| = 1$, alors $q_j(u) = 0$ (la réciproque est fausse). On peut en déduire que le test d'homogénéité d'un sommet v peut être fait au moment où l'on essayera de développer v (au lieu d'être fait lors du développement de son père). Ainsi, certains tests seront évités (ceux de tous les sommets de P' lorsqu'un backtracking intervient), tout en étant sûr de ne pas passer à côté de sommets terminaux sans les voir (puisqu'ils correspondent au connecteur minimal).

La relation suivante est plus importante à exploiter :
Si pour u et $j \in \Phi(u)$: $q_j(u) = 0$, alors $\forall v \in \Gamma_j(u)$ tel que $j \in \Phi(v)$: $q_j(v) = 0$.
En d'autres termes, si une caractéristique X_j est nécessaire dans un sous-système S/u, alors ou bien elle sera évaluée, ou bien elle restera nécessaire dans chacun des sous-systèmes S/v, v successeur de u. Une procédure peut donc être écrite pour accélérer la recherche dans ce cas : lors de l'examen du sommet u, on effectuera un développement en profondeur de tous les connecteurs (dans un ordre quelconque) correspondant à ces caractéristiques nécessaires, et cela sans incrément des estimations. Relativement à la racine de H, cette procédure peut être appliquée non seulement pour les caractéristiques nécessaires dans tout S, mais aussi pour celles à coût nul et non redondantes. Notons que cette procédure est en fait une décomposition (algorithmiquement simple) du système initial en plusieurs sous-systèmes qui seront optimisés les uns indépendamment des autres.

Il est possible d'avoir pour une caractéristique X_j : $q_j(u) \neq 0$ et $q_j(v) = 0$ pour certain $v \in \hat{\Gamma}_j(u)$. Dans un cas avantageux, on a la proposition suivante :

Proposition 3.6. : S'il existe dans H un SL Λ tel que pour tout connecteur $\Gamma_j(u) \in \Lambda$ on ait : $q_j(u) = 0$, alors Λ est un arbre optimal et, quel que soit le paramètre ε fourni en entrée à AO_ε, cet algorithme effectuera une simple recherche en profondeur dans H (sans backtracking) et fournira, en sortie, la solution Λ, ainsi que sa garantie d'optimalité ($\varepsilon' = 0$).

Preuve : L'optimalité de Λ résulte immédiatement des faits que h est minorante et $\Psi(\Lambda) = h(u_0)$, u_0 racine de H. L'algorithme AO_ε ne développe que les sommets appartenant à Λ et ne "backtrack" pas, car tout au long de la recherche, on a $F \leqslant F'$ (premier cas de la preuve du lemme 2.14.). □

Une autre relation intéressante peut être exploitée :

$$q_j(u) = \sum_{v \in \Gamma_i(u)} q_j(v) \quad ; \quad i \neq j$$

Pour l'un des z_i successeurs de u sur $\Gamma_i(u)$, la quantité $q_j(v)$ se déduit simplement par soustraction. Ceci est simplifié par le développement des sommets de même rang dans l'ordre transverse.

Finalement, certaines améliorations de l'algorithme qui dépendent étroitement du choix d'une structure de donnée particulière, seront présentées au chapitre 4 avec la description d'un exemple d'implémentation.

Discutons, à présent, quelques variantes dans les stratégies d'exploration et de backtracking d'AO_ε. Comme il a été mentionné précédemment (section 2.7.), il n'est pas nécessaire de prendre $g(u) = \min_j \{ g_j(u) \}$ et $\hat{j}$ l'indice de minimum. On peut poursuivre la recherche sur tout connecteur conservant l'acceptabilité de Λ, i.e. prendre $\hat{j}$ dans l'ensemble $\{ j \in \Phi(u) \mid F + g_j(u) \leqslant (1+\varepsilon)F' \}$.

L'heuristique h_c pour guider le choix de $\hat{j}$ peut être construite à partir du degré de développement de $\Lambda_j(u)$, SL de racine u sur le connecteur $\Gamma_j(u)$.

Il est beaucoup plus intéressant de poursuivre la recherche sur un connecteur dont tous les successeurs sont développés, que sur un connecteur dont tous les successeurs sont encore pendants : non seulement il y aura moins de développements à faire par la suite, mais surtout les estimations g(v) des successeurs développés seront prises en compte dans $g_j(u)$, entraînant ainsi moins de risque de backtracking ultérieur . D'où une définition possible de h_c :

- si v est un sommet terminal : $h_c(v) = 0$;
- si v est un sommet pendant de rang k : $h_c(v) = n - k$;
- si v est un sommet développé : $h_c(v) = \frac{1}{z_{\hat{j}(v)}} \sum_{w \in \Gamma_{\hat{j}}(v)} h_c(w)$.

Lors du développement de u, on prendra pour $\hat{j}(u)$ le connecteur acceptable qui minimise $h_c(u)$. On vérifie que cette heuristique choisira toujours le connecteur correspondant à un SL complet s'il en existe (SL sans sommet pendant), sinon elle sélectionnera celui dont le rang moyen des sommets pendants est maximal. Remarquons que les rangs n'y interviennent que par leurs valeurs relatives.

L'inconvénient majeur de cette heuristique est le coût algorithmique supplémentaire nécessaire à son actualisation et à sa mise à jour.

A priori, l'éventuel bénéfice de la stratégie risque d'être remis en cause par ce coût de mise en oeuvre. On peut alors se rabattre sur une heuristique beaucoup moins riche, mais n'entraînant pas de calcul supplémentaire : un sommet v sera dit complet s'il est soit terminal, soit racine d'un SL $\Lambda_{\hat{j}}(v)$ complet ; et on poursuivra le développement d'un sommet u par le connecteur acceptable ayant un nombre maximum de descendants complets.

De même qu'il n'est pas nécessaire, lors d'un développement, de poursuivre sur le connecteur minimal ; il n'est pas nécessaire lors d'un backtracking de revenir au meilleur SL de H. Si le SL Λ en cours d'exploration a cessé d'être acceptable, l'actualisation ascendante, en modifiant les connecteurs $\hat{j}$, changera progressivement Λ à partir de ses sommets de plus grand rang. Une mise à jour décrémentale de l'estimation F du coût de Λ peut avoir lieu à la suite de l'actualisation de chaque sommet de Λ ; et il est possible d'arrêter l'actualisation globale de H dès que le nouveau Λ est redevenu acceptable relativement à F'.

En effet, les nouveaux sommets de Λ qui l'ont rendu non acceptable ne remettent pas en cause le minorant F' (lequel ne peut qu'augmenter). Par ailleurs, l'estimation de tous ces sommets de Λ a été comptée dans F (par DEVELOPPEMENT). Si l'actualisation d'un sommet u change son connecteur minimal $\hat{j}$ de l'indice $\hat{j}_1$ à l'indice $\hat{j}_2$, cela veut dire que $\Lambda\, \hat{j}_2(u)$ a une meilleure estimation que $\Lambda\, \hat{j}_1(u)$. L'estimateur global F peut alors être décrémenté de :

$$F \leftarrow F - \left[g_{\hat{j}_1}(u) - g_{\hat{j}_2}(u) \right] ;$$

les valeurs $g_j(u)$ étant celles <u>après</u> actualisation de u.

L'inconvénient de cette stratégie est de conduire à des "backtracking" courts et nombreux : on revient à la première bifurcation acceptable en se contentant de petites modifications locales sur Λ. Appliquée en début d'exploration sur un H peu développé, cette stratégie sera bloquée par une valeur de F' relativement faible. En conduisant l'algorithme à s'obstiner laborieusement sur des améliorations d'autant plus mineures que l'incidence sur le coût global diminue avec le rang, elle risque d'être le plus souvent pénalisante. Par contre, en fin d'exploration, lorsque le Λ obtenu est presque complet et que seules quelques "retouches" sur certaines branches sont nécessaires pour converger, cette stratégie est nettement avantageuse relativement à un "backtracking" long qui repartirait à la racine u_o en gaspillant l'information acquise. La difficulté réside, bien entendu, dans l'appréciation des phases début et fin de l'exploration.

Remarquons que la valeur (relative) de F' est déterminante dans cette appréciation. On peut profiter du fait que l'incidence sur le coût global est d'autant plus grande que le rang d'un sommet est faible pour essayer d'augmenter F' au prix d'un léger investissement algorithmique. Quelques variantes de la stratégie persévérante, discutée au chapitre précédent, pourraient être les suivantes :

- développer systématiquement au <u>début</u> de l'algorithme tous les sommets de H de rang inférieur ou égal à k (k faible : 2 ou 3) ;

- développer au début de l'algorithme l'hyperlatticiel H le long des m meilleurs connecteurs issus de sa racine u_o, puis récursivement, le long des m meilleurs de leurs descendants, et ainsi de suite jusqu'au rang k, avec $m < n - k$;

- même stratégie pour démarrer l'algorithme que la précédente, mais m n'étant plus un paramètre constant : on développe le long du meilleur connecteur et de tous les connecteurs dont l'estimation ne s'écarte pas de celle du meilleur de plus de α % .

Toutes ces stratégies permettraient de commencer l'exploration sur un hyperlatticiel à un stade avancé de développement et avec une valeur relative de F' importante.

Une approche différente consisterait à essayer d'augmenter F' en fin d'exploration, lorsqu'une solution acceptable est déjà connue, et cela dans le but d'améliorer le majorant $\mathcal{E}'$. On adoptera cette approche, en particulier, lorsque le schéma d'approximation est utilisé, non pas avec une borne ϵ sur l'acceptabilité de la solution requise, mais plutôt avec une borne sur les ressources de calcul disponibles pour la recherche (on répond à une demande du type : que pourrais-je avoir de mieux pour s secondes de CPU ?).

Une procédure qui ne s'arrêterait qu'à l'épuisement des ressources pourrait être la suivante :

- effectuer une recherche en profondeur dans H ($\mathcal{E} = \infty$) qui fournirait une première solution t et un majorant $\mathcal{E}'$;

- itérer tant qu'il reste des ressources :
  - Si les ressources encore disponibles sont faibles (par exemple, inférieures à celles consommées pour atteindre t), on les exploitera dans l'amélioration de $\mathcal{E}'$ par une recherche dans H selon la stratégie opportuniste ;
  - Sinon (ressources encore suffisantes), on tentera une recherche $\mathcal{E}$- admissible, en utilisant le coût de la meilleure solution connue comme fonction d'élimination des sommets de H, et avec $\mathcal{E} = \mathcal{E}' / \alpha$ (α est fonction des ressources disponibles par rapport à ceux précédent utilisées).
  - Fin.

Il est bien entendu possible de formuler le problème avec les deux contraintes d'acceptabilité et de ressources. De même, il est possible de combiner

plusieurs des stratégies discutées ici. Par exemple, on peut procéder à une des variantes de développement partiel systématique de H pour initialiser l'algorithme ; faire des backtracking "longs" en début d'exploration, puis des backtracking "courts" en fin de recherche lorsque le SL exploré est presque complet, et finalement, lorsqu'une solution acceptable est atteinte, terminer par une stratégie opportuniste pour améliorer ϵ .

Certaines des stratégies proposées ont été implémentées. Le chapitre 4 présente quelques résultats empiriques sur leur utilisation.

Terminons cette section par une analyse de la complexité maximale de schéma d'approximation obtenu. On se rappelle que dans le cas général, sur un hyperlatticiel fini de N sommets, AO^* et AO ont, dans le pire cas, la même complexité en $O(N^2)$. Ici, H possède au maximum $N = \prod_{i=1}^{n} (1 + z_i)$ sommets, et donc le schéma d'approximation est au plus en $O\left(\left[\prod_{i=1}^{n} (1 + z_i)\right]^2\right)$. Améliorons ce majorant en exploitant la structure particulière du problème.

Dans le pire cas, l'algorithme développe tous les $\prod_{i=1}^{n}(1 + z_i)$ sommets de H, et chaque sommet développé entraîne l'actualisation de tous ses ascendants. Bien que le coût de développement ou d'actualisation d'un sommet diminue avec son rang (si v est de rang k, le nombre d'itérations sur v dans DEVELOPPER ou ACTUALISER est majoré par $\Phi(v) < n - k$), continuons à admettre un même coût unitaire (maximal) pour une quelconque de ces deux opérations, considérées comme élémentaires. Le nombre de sommets de H au rang k est :

$$\sum_{i_1=1}^{n-k+1} \sum_{i_1<i_2<n-k+2} \cdots \sum_{i_{k-1}<i_k\leq n} z_{i_1} \cdots z_{i_k}$$

(Toutes les façons de choisir k coordonnées explicites parmi n, et pour chaque choix, toutes les combinaisons possibles des valeurs de ces coordonnées).

Un sommet v au rang k possède :

C^1_k pères au rang k-1 (choix de 1 coordonnée explicite dans v parmi k) ;

C^2_k ascendant au rang k-2 (choix de 2 coordonnées explicites dans v parmi k) ;

....

au total, le nombre des ascendants de v est : $\sum_{i=1}^{k} C^i_k = 2^k - 1$.

Le développement de v et l'actualisation de tous ses ascendants entraînent donc 2^k opérations. La complexité maximale de l'algorithme est donc :

$$\sum_{k=0} \left[\sum_{i_1=1}^{n-k+1} \sum_{i_1<i_2<n-k+2} \sum_{i_{k-1}<i_k\leq n} z_{i_1} \cdots z_{i_k} 2^k \right]$$

Cette expression est égale à : $\prod_{i=1}^{n} (1 + 2z_i)$.

(Preuve par récurrence :

$$\prod_{i=1}^{n} (1 + 2 z_i) = (1 + 2z_n) \times \prod_{i=1}^{n-1} (1 + 2 z_i)$$

$$= (1 + 2z_n) \times \sum_{k=0}^{n-1} \left[\sum_{i_n=1}^{n-k-2} \cdots \sum_{i_{k-1}<i_k\leq n-1} z_{i_1} \cdots z_{i_k} \cdot 2^k \right]$$

$$= 1 +$$

$$+ \sum_{i=1}^{n-1} 2 z_i \qquad + 2z_n$$

$$+ \sum_{i_1=1}^{n-2} \sum_{i_1<i_2\leq n-1} 4 z_{i_1} z_{i_2} \qquad + 2z_n \times \sum_{i=1}^{n-1} 2z_i$$

$$\cdots \qquad \cdots$$

$$+ \prod_{i=1}^{n-1} z_i \cdot 2^{n-1} \qquad + 2z_n \times \sum_{j=1}^{n-1} \prod_{\substack{i=1 \\ i\neq j}}^{n-1} z_i 2^{n-2}$$

$$+ \qquad + 2z_n \times \prod_{i=1}^{n} z_i 2^{n-1}$$

□)

On retrouve, dans le cas de n caractéristiques binaires, le résultat de (14 : sur AO* : complexité en $O(5^n)$.

On ne peut malheureusement améliorer danvantage l'ordre de complexité du majorant précédent, car il constitue une borne effectivement atteinte par l'algorithme comme le montre l'exemple suivant :

Exemple 3.3. : Soit le système constitué de n caractéristiques binaires ; de 2^{n-1} règles chacune correspondante à un 0-cube unique contenant un nombre impair de coordonnées égales à 1 et les autres à 0 ; et de 2^{n-1} relations de dépendance :

tous les 0-cubes ayant un nombre pair de coordonnées à 1. La distribution de probabilité est uniforme, et le coût de X_i est : $\rho_i = 1 + \alpha.i$; $\alpha < 1/2^n$. Pour $1/\varepsilon \geq (n-1) \times 2^{n-2} - 1$, on vérifiera que la symétrie de S conduit le schéma d'approximation à développer tous les sommets de H de rang $k \leq n - 2$, et pour chaque sommet développé, à actualiser tous ses $(2^k - 1)$ ascendants. (Un même coût unitaire pour les n caractéristiques ne conduirait pas à chaque fois l'actualisation jusqu'à la racine de H : on changerait sur certains sommets u de connecteur $j(u)$, mais pas de minimum de $g_j(u)$).

En effet, il est simple de voir que toutes les règles figurent dans les ensembles consensus de toutes les caractéristiques, et cela pour tout sous-système S/u, $u \notin D$. Toutes les caractéristiques sont donc informationnellement redondantes et, dans tout état, (n - 1) caractéristiques sont nécessaires et suffisantes pour accéder à une décision. L'arbre optimal possède 2^{n-1} feuilles, chacune correspondant à un chemin testant successivement $X_1, X_2, \ldots, X_{n-1}$.

Remarque : Cet exemple extrême montre que le majorant obtenu est une borne, mais il n'est pas illustratif du comportement général de l'algorithme. En fait, il n'est même pas illustratif d'un comportement exponentiel : l'algorithme est bien en $O(5^n)$, mais vu la taille des données d'entrée, $N = 2^n$, il est polynômial en $O(N^{\log 5/\log 2}) = O(N^{2.32})$.

3.2.4. Post-optimisation : améliorations d'un arbre de décision

Certaines améliorations peuvent être apportées à l'arbre de décision acceptable t, représentant le système S, obtenu par le schéma d'approximation précédent. L'une d'elles concerne la transforamtion de cet arbre en latticiel : les X_j ne définissent qu'un recouvrement de A (et non une partition), et si t a plus de $a = |A|$ feuilles, il possède nécessairement des sous-arbres identiques ; l'élagage des duplications conduira à un latticiel de a feuilles.

Une procédure peu sophistiquée consisterait à associer à chaque noeud de t un code représentant le sous-arbre dont ce noeud est la racine. Par exemple, à une feuille contenant l'action A_k, on associe la liste (A_k), et à un noeud contenant la caractéristique X_i, la liste $(X_i, L_o, \ldots, L_{z_i-1})$; $L_o, \ldots, L_{z_i-1}$ étant les

listes correspondantes aux branches issues de X_i (représentation parenthésée d'un arbre). Une exploration ascendante qui coderait tous les noeuds de l'arbre, serait suivie d'une recherche descendante dans l'ordre des rangs. Le code du noeud v examiné est comparé à celui des noeuds qui le précèdent dans la recherche et, en cas d'égalité avec le code de y, le sous-arbre $\hat{\Gamma}(v)$ est élagué, et la branche $(\Gamma^{-1}(v), v)$ est transformée en $(\Gamma^{-1}(v), y)$. Cette procédure est de complexité quadratique en $|X|$.

Est-il possible, en modifiant $AO\varepsilon$, de construire directement la structure décisionnelle représentant S sous forme d'un latticiel ?

La réponse est, a priori, affirmative : on explore un hyperlatticiel dans lequel chaque sommet ne représente plus un r-cube unique, mais une union de r-cubes (non nécessairement adjacents). Deux termes u et v pourront être unis dans le même sommet ssi les sous-systèmes S/u et S/v sont identiques : mêmes ensembles de caractéristiques et de règles (X/u = X/v et R/u = R/v), mais aussi mêmes distributions de probabilité sur les termes de S/u et ceux de S/v (l'absence de cette dernière condition ne garantirait pas que l'algorithme trouverait les mêmes sous-arbres à partir de S/u et de S/v). Non seulement cette condition nécessaire est beaucoup plus difficile à tester que l'identité des sous-arbres, mais, de plus, elle fait intervenir tous les sous-systèmes explorés par l'algorithme (leur nombre est supérieur à celui des sous-arbres de la solution fournie). Finalement, ces regroupements ne garantissent même pas l'absence de sous-latticiels identiques dans le latticiel solution.

On se contentera donc d'améliorations a postériori sur l'arbre, et cela d'autant plus qu'il existe pour le faire des procédures plus performantes que celle donnée précédemment en exemple. Myers(166) en propose plusieurs qui s'apparentent aux techniques de post-optimisation utilisées dans certains compilateurs. Les manipulations d'un arbre qui y sont proposées vont jusqu'au"hissage des actions" : décomposition d'une action globale en ses composantes élémentaires, et si une composante est commune à toutes les feuilles d'un sous-arbre, elle est exécutée dès la racine de ce sous-arbre.

On ne discutera pas davantage ici ce type d'améliorations à postériori d'arbre de décision, car comme on l'a montré précédemment, ces transformations ne

modifient pas le coût du PDF sur l'arbre, mais affectent seulement le coût de mémorisation de sa structure, critère peu intéressant dans notre optique. Développons plutôt une transformation à postériori d'un arbre t qui affecterait le coût $\Psi(t)$.

Si t contenait un noeud v dont toutes les branches mènent à des sous-arbres identiques, alors ce noeud et tous ses sous-arbres, sauf un, pourraient être supprimés, et le père de v serait relié au sous-arbre restant. L'arbre résultant t' aurait comme coût :

$\Psi(t') = \Psi(t) - \mu(v) * \rho_i$; avec ρ_i coût de $X_i = \mathcal{V}(v)$.

Cependant, une telle situation ne peut se produire dans l'arbre t fourni par AO_ϵ : $X_i = \mathcal{V}(v)$ serait une caractéristique complètement redondante dans S/v (cas (i) de la proposition 3.5., les sous-systèmes $S/(v/i \leftarrow 0), \ldots, S/(v/i \leftarrow z_i - 1)$ étant identiques) et donc éliminée par l'algorithme lors du développement du sommet v de H. Mais, la redondance dans S/u est une propriété locale qui n'est détectée par l'algorithme que si u (ou $u' \in \hat{\Gamma}(u)$) est développée. Il n'est pas exclu qu'un arbre t' fortement équivalent à t ne contienne un noeud avec une caractéristique complètement redondante. t' définissant la même partition sur $(\sum_{i=1}^{n} z_i)$ que t, a le même coût que t. Une simplification de t' conduirait donc à la même réduction de coût que ci-dessus.

Exemple 3.4. : L'arbre t (figure 3.4.) définit sur $\{0, 1\}^4$ la partition $\pi = (0\ 0\ \varphi\varphi), (0\ 1\ \varphi\ 0), (0\ 1\ \varphi\ 1), (1\ 0\ \varphi\varphi), (1\ 1\ 0\ \varphi), (1\ 1\ 1\ \varphi)$. L'arbre t' est fortement équivalent à t, et possède un noeud dont toutes les branches sont identiques. Notons que la caractéristique correspondante X_1, complètement redondante dans le sous-système $S/(\varphi\ 0\ \varphi\ \varphi)$, n'est pas redondante dans le système initial.

Il est simple de voir que l'existence d'une redondance dans un arbre t' fortement équivalent à t, implique que dans la partition $\pi = (l_1, \ldots, l_q)$ commune à t et t', plusieurs termes correspondant à la même action sont adjacents (selon la caractéristique redondante). La réciproque de cette implication est fausse. Pour mettre en oeuvre la simplification mentionnée, on cherchera les adjacences des termes de π qui conservent une partition π' ($\pi \subset \pi'$) définissable

par un arbre.

Insistons sur le fait que pour rester dans la même famille d'arbres fortement équivalents, on s'interdit de décomposer un bloc de π .

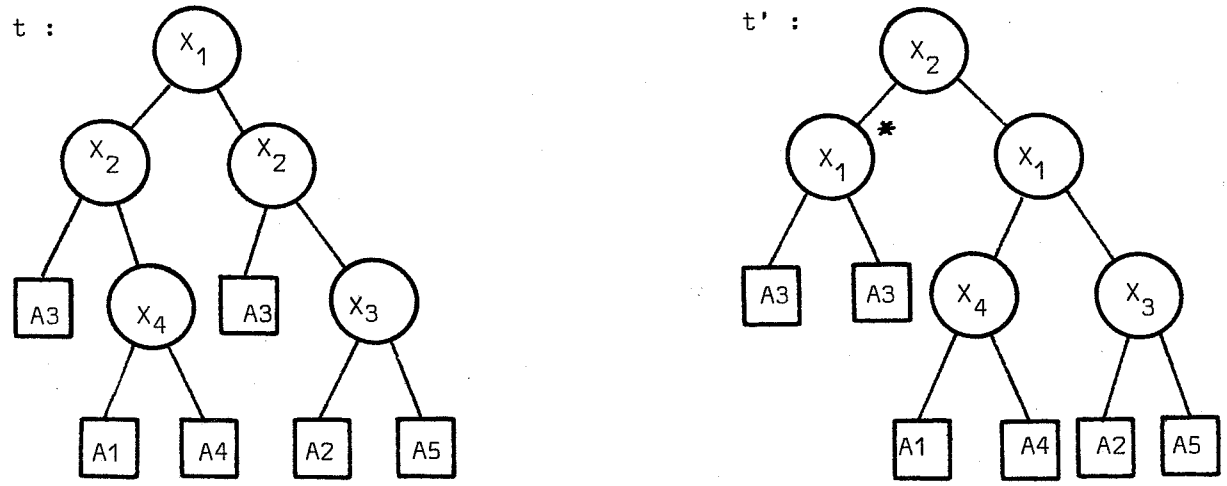


FIGURE 3.4.

Une procédure, admettant comme entrée l'arbre t fourni par AO_ε , pourrait être la suivante :

Procédure SIMPLIFICATION

1. Déterminer toutes les adjacences possibles entre blocs de la partition $\pi = (l_1, \ldots, l_q)$ définie par t qui correspondent à la même action.

2. Si aucune adjacence n'est trouvée, alors t ne peut être amélioré ; arrêt de la procédure en donnant en sortie t' = t.

3. Sinon, soit $l'_{i_1} = \bigcup_{1 \leq k \leq \beta i_1} l_{j_k}$; l'_{i_2} ; ... l'_{i_r} les regroupements des blocs pour toutes les adjacences trouvées, ceux de l'_{i_k} étant adjacents par rapport à la caractéristique X_{i_k}.

3.1. On construit l'arbre t' fortement équivalent à t en prenant comme racine de t' la caractéristique X_j tel que :

(i) pour tout bloc l_k de π : $l_k(j) \neq \varphi$; et

(ii) la somme sur toutes les adjacences trouvées sur X_j :

$\sum_{j=i_k} \mu(l'_{i_k}).p_j$ est minimale.

(Il existe au moins une X_j vérifiant (i), car π est définissable par un arbre ; (ii), qui minimise la perte à gagner dans le choix de X_j, conduira aussi à prendre une caractéristique qui ne figure dans aucun regroupement si elle existe).

3.2. On reprend récursivement la construction de t' sur chacune des branches de X_j, en considérant pour la $i^{ème}$ branche la sous-partition de π: $\{l_k \in \pi \mid l_k(j) = i\}$; et tous les regroupements l_{i_k} tel que $i_k \neq j$. Une branche est transformée en feuille dès que tous les blocs de sa partition correspondent à la même action.

On vérifie simplement que :

- l'arbre t' résultant est optimal dans la classe des arbres fortement équivalents à t ;

- la procédure est au plus en $O(nq^2z)$, avec $z = \max_i \{z_i\}$.

Notons, pour finir, qu'une amélioration éventuelle du coût de l'arbre de décision pourra être répercutée sans difficulté sur le majorant ε' de son nouveau degré d'optimalité : $\varepsilon' = \varepsilon - (\Psi(t) - \Psi(t'))/\Psi(t')$.

°0°

3.3. GENERALISATIONS PARAMETRIQUES

Cette section aborde le problème d'optimisation des processus décisionnels pour les autres modèles de coûts introduits en (1.4.). L'algorithme utilisé continuera à être un schéma d'approximation du type AO_ϵ . Aussi, pour chaque modèle, on développera une fonction heuristique h dont on établira les propriétés et on discutera les éventuelles modifications à faire dans l'algorithme de base pour l'adapter au cas particulier traité.

3.3.1. MODELES DES COUTS CONDITIONNELS ET DES COUTS DEPENDANTS

La généralisation du modèle des coûts constants aux coûts conditionnels sera très simple. Dans ce modèle, le coût $\rho(j/i_1, \ldots, i_r)$ de la caractéristique X_j dépend des caractéristiques $X_{i_1}, \ldots, X_{i_r}$ évaluées avant X_j.

On vérifie facilement que la structure de coût de H est :

$$K(u,j) = \rho(j/u) \cdot \mu(u)$$

avec : $\rho(j/u) = \rho(j/i_1, \ldots i_r)$ pour $\{i_1, \ldots i_r\} = \{1, \ldots, n\} - \Phi(u)$.

Notons le coût minimum d'une caractéristique X_j par :

$$\bar{\rho}_j = \min\{\rho(j/i_1, \ldots i_r) \mid \{i_1, \ldots, i_r\} \subset \{1, \ldots, n\},\ 0 \leq r \leq n\} ;$$

et définissons la fonction h

$$h(u) = \sum_{i \in \Phi(u)} \bar{\rho}_i \, (\mu(u) - q_i(u))$$

Proposition 3.7. : h(u) est une évaluation heuristique monotone et coïncidante du coût d'un arbre optimal représentant S/u.

Preuve :

$$\sum_{v \in \Gamma_j(u)} h(v) = \sum_{v \in \Gamma_j(u} \sum_{i \in \Phi(u)} \bar{\rho}_i \, (\mu(u) - q_i(u))$$

$$\sum_{v \in \Gamma_j(u)} = \sum_{\substack{i \in \Phi(u) \\ i \neq j}} \bar{\rho}_i \sum_{v} \left(\mu(v) - q_i(v) \right)$$

$$= \sum_{i \neq j} \bar{\rho}_i \left(\mu(u) - q_i(u) \right) = h(u) - \rho_j \left(\mu(u) - q_j(u) \right)$$

$$K(u,j) + \sum_{v \in \Gamma_j(u)} = h(u) + \bar{\rho}_j q_j(u) + \mu(u) \left[\rho(j/u) - \bar{\rho}_j \right]$$

par définition $\bar{\rho}_j \leq \rho(j/u)$, donc h est bien monotone. La preuve de la coïncidence de h est identique à celle du modèle des coûts constants. □

On remarque que l'incrément de coût, après le développement d'un sommet u, est :

$$f_j(u) = \bar{\rho}_j - q_j(u) + \mu(u) \left[\rho(j/u) - \bar{\rho}_j \right]$$

Cet incrément n'est pas plus complexe à déterminer que dans le cas général. Même la simplification qui consistait à prendre $f_j(u) = 0$ dès que $|\Phi(u)| = 1$ demeure valide ici ($q_j(u) = 0$ et $\rho(j/u) = \bar{\rho}_j$).

Par ailleurs, le traitement des redondances des caractéristiques est le même que dans le modèle de base, à condition de supposer une "cohérence" dans les distributions de coûts, i.e., que le coût d'évaluation à priori d'une caractéristique X_i n'est par inférieur à la diminution du coût des autres caractéristiques due à la connaissance de X_i.

Dans le modèle des coûts dépendants, on dispose de k sous-ensembles de caractéristiques $\mathcal{C}_1, \ldots, \mathcal{C}_k$ de coûts respectifs $\rho_1, \cdots, \rho_k$. Une caractéristique $X_i \in \mathcal{C}_j$ coûte ρ_j s'il n'existe aucune autre caractéristique déjà évaluée et appartenant à $\mathcal{C}_j$, sinon le coût d'évaluation de X_i est nul.

On peut considérer ce modèle comme un cas particulier du précédent. Néanmoins, s'il y a un nombre important de dépendances, $\vec{\rho}_i = 0$ sur presque toutes les caractéristiques.

L'heuristique h sera souvent nulle, et donc peu efficace.

Sur un plan pratique, le modèle des coûts dépendants correspond, en général, au cas où chaque $\mathcal{C}_j$ reflète un logiciel calculant simultanément plusieurs X_i. On peut admettre que la répartition des n caractéristiques de X est faite de façon à ce que $\mathcal{C}_1, \ldots, \mathcal{C}_k$ en soit une partition. Il est alors possible par regroupement des X_i communs à un même $\mathcal{C}_j$, de passer d'un système de n caractéristiques à un système de $k < n$ macro-caractéristiques.

Sauf, dans le cas où la dépendance en coût est étroitement couplée à la dépendance structurelle des caractéristiques (auquel cas, cette redéfinition de X en $\{\mathcal{C}_1, \ldots, \mathcal{C}_k\}$ s'impose) ; un tel regroupement sera algorithmiquement très lourd. Il nécessitera l'éclatement en composantes élémentaires de tout terme u, tel que Φ (u) ne corresponde pas aux caractéristiques d'un même bloc $\mathcal{C}_j$.

Toujours en supposant que $\mathcal{C}_1, \ldots, \mathcal{C}_k$ est une partition de X, on peut adopter une autre approche qui consiste à associer à $X_i \in \mathcal{C}_j$ un coût moyen $\bar{\rho}_i = \rho_j / |\mathcal{C}_j|$ et à utiliser l'estimateur h précédent avec cette définition de $\bar{\rho}_i$. L'estimateur n'aura plus les propriétés de minoration et de coïncidence. Il peut, néanmoins, être utilisé dans l'optique de la recherche d'une approximation non bornée, en modifiant l'un des algorithmes A1 ou SOUS-LATTICIEL de façon à ce que le choix de X_i, pour un certain noeud de l'arbre, entraîne le développement de toutes les branches issues de i par des sous-arbres contenant toutes les autres caractéristiques communes au même bloc que X_i.

On retient, en conclusion pour ce modèle des coûts dépendants, que la seule approche satisfaisante correspond à la redéfinition des caractéristiques sur les bases des dépendances structurelles et des dépendances de coût lorsque ces deux dépendances sont suffisamment corrélées.

3.3.2. MODELE DES COUTS VARIABLES

Ici, le coût d'évaluation de X_j dépend de la valeur de cette caractéristique. Sur un état e tel que $X_j(e) = k$, ce coût est : $\rho(j,k)$.

La distribution correspondante sur les connecteurs de H est :

$$K(u,j) = \sum_{0 \leq k \leq \gamma_j} \rho(j, k) \times \mu(u \mid j \leftarrow k).$$

On vérifie très simplement que cette distribution conduit bien au coût d'un arbre tel que défini en 1.4.6. :

$$\psi(t) = \sum_{\substack{v \in V \\ \mathcal{V}(v) = C_j}} \sum_{0 \leq k < z_j} p(j,k) \times \mu\left(\Gamma_k(v)\right)$$

Si on généralise l'heuristique du modèle des coûts constants sur la base des K(u,j) ; coûts moyens des caractéristiques en un noeud u, i.e., en utilisant comme estimation la valeur de :

$$\sum_{j \in \Phi(u)} \sum_{0 \leq k < z_j} p(j,k) \times \mu(u \mid j \leftarrow k) \times \left[\mu(u) - q_j(u)\right]$$

on n'aura plus un estimateur minorant. En fait, il est nécessaire de revenir à la définition initiale de h, qui est l'espérance de coût nécessaire en un sommet u, ce que ne représente pas la formule précédente.

Soit $\overline{C}_i = \{ v \subset u \in S \mid v \notin C_i \}$, i.e., l'ensemble des termes u de S tel que $u(i) \neq \varphi$, et que u ne donne lieu à aucun regroupement sur X_i ; et des sous-termes $v \subset u \in S$ tel que v n'est inclus dans aucun des regroupements auxquels a donné lieu u. Par définition de $\overline{C}_i$, $\forall e \in E$ tel que $(X_1(e),\ldots,X_n(e) \subset v \in \overline{C}_i$, et pour tout arbre t représentant S, le processus décisionnel PDF (t,e) nécessitera l'évaluation de la caractéristique X_i.

On définit :

$q'_i = \sum_{i \in \overline{C}_i(u)} \mu(v) = 1 - q_i$: probabilité de se trouver dans un état où l'évaluation de X_i est nécessaire ;

$q'_i(u) = \sum_{v \in \overline{C}_i} \mu(v \cap u) = \mu(u) - q_i(u)$: probabilité jointe d'avoir l'événement précédent, et que cet état corresponde à un des sous-termes de u.

L'espérance de coût nécessaire en u est alors :

$$h(u) = \sum_{i \in \Phi(u)} \sum_{0 \leq k < z_i} p(i,k) \cdot q'_i(u \mid i \leftarrow k)$$

Proposition 3.8. : h est une évaluation monotone et coïncidente.

Preuve : On a $q'_i(u/i \leftarrow k) = \sum_{v \in \Gamma_j(u)} q'_i(v/i \leftarrow k)$; $i \neq j$.

$$\sum_{v \in \Gamma_j(u)} h(v) = \sum_{v} \sum_{i \neq j} \sum_{0 \leq k < z_i} p(i,k)\ q'_i(v \mid i \leftarrow k)$$

$$= \sum_{i \neq j} \sum_{k} \sum_{v} p(i,k)\ q'_i(v \mid i \leftarrow k)$$

$$= \sum_{i \neq j} \sum_{k} p(i,k) \cdot q'_i(u \mid i \leftarrow k)$$

D'où : $$K(u,j) + \sum_{v \in \Gamma_j(u)} h(v) = h(u) + \sum_{0 \leq k < z_j} p(j,k) \left[\mu(u \mid j \leftarrow k) - q'_j(u \mid j \leftarrow k) \right]$$

La monotonie de h en résulte simplement.

Coïncidence : si u est terminal, plus aucune caractéristique n'est nécessaire au processus décisionnel, et $\forall i \in \Phi(u) : \overline{C}_i \cap u = \emptyset$, donc :

$q'_i(u) = 0$ et $h(u) = 0$. □

Comme on l'a remarqué précédemment, $\overline{C}_i$ n'est pas aussi simple à déterminer que C_i. En utilisant l'algorithme RECOUVREMENT, on procèdera par différence par rapport aux termes mis dans C_i : si $u \in S$ contient un sous-terme v qui donne lieu à un recouvrement sur X_i, on supprimera u de S en le remplaçant par (u-v) ; $\overline{C}_i$ sera l'ensemble des termes restant dans S en fin d'itération sur X_i. On obtient une procédure de complexité exponentielle en pire cas, car la différence (u-v) entraîne le remplacement d'un terme unique u par $\prod_{i \in \Phi(u) - \Phi(v)} z_i$ termes. Cette procédure demeure utilisable dans les cas, assez fréquents, où il y a peu de recouvrements autres que ceux contenus dans la formulation initiale de S (si C_i ne contient que les $u \in S$ tel que $u(i) = \varphi$. $\overline{C}_i$ en sera le simple complémentaire).

Remarque : L'ensemble C_i continuera d'être utile pour l'élimination des caractéristiques complètement redondantes, le test ne se faisant néanmoins que sur les X_i telles que $\sum_{v \in C_i} \mu(v \cap u) = 0$.

Si la détermination des $\overline{C}_i$ s'avère très difficile, on peut préconiser dans une approche d'approximation non bornée, l'utilisation de l'heuristique h du modèle de base avec les coûts moyens à priori des caractéristiques :

$$\sum_{0 \leq k < z_j} p(j,k) \cdot \mu(j \leftarrow k).$$

Combinaison des modèles des coûts variables et conditionnels

Si $\rho(j, k/i_1, \ldots, i_r)$ est le coût de X_j pour un état e tel que $X_j(e) = k$, et lorsque $X_{i_1}, \ldots, X_{i_r}$ ont été évalués, on prendra :

$$\bar{\rho}(j,k) = \min\{\rho(j, k \mid i_1, \ldots i_r) \mid \{i_1, \ldots i_r\} \subset \{1, \ldots n\}\} \text{ ; et}$$

$$h(u) = \sum_{i \in \Phi(u)} \sum_{0 \leq k < z_i} \bar{\rho}(i,k) * q'_i(w / i \leftarrow k)$$

L'incrément de coût, après le développement d'un sommet u, sera :

$$g_j(u) = \sum_{0 \leq k < z_j} [\rho(j,k \mid u) * \mu(u \mid j \leftarrow k) - \bar{\rho}(j,k) * q'_j(u / j \leftarrow k)]$$

La monotonie de h ($g_j(u) \geq 0$) et sa coïncidence se vérifient très simplement.

3.3.3. MODELE DES COUTS D'ACCES

On rappelle que dans ce modèle (I_1, I_2) est une partition de $\{1, \ldots, n\}$ telle que :

- pour $i \in I_1$, la caractéristique X_i correspond à un logiciel mis en mémoire principale ; elle y occupe l'espace ζ_i dans un espace total disponible r ;

- pour $i \in I_2$, la caractéristique X_i est en mémoire secondaire, et son évaluation entraîne, en plus du coût ρ_i, un coût d'accès ξ_i.

On cherche un arbre t représentant S et une partition (I_1, I_2) de $\{1, \ldots, n\}$, de façon à minimiser le coût moyen total de processus décisionnel :

$$\psi(t) = \sum_{i=1}^{n} \rho_i \sum_{X_i = \mathcal{V}(v)} \mu(v) + \sum_{i \in I_2} \xi_i \sum_{X_i = \mathcal{V}(v)} \mu(v)$$

en respectant la contrainte $\sum_{i \in I_1} \zeta_i \leq r$.

Dans une première approche, on décompose le problème en :

i) recherche de l'arbre t en supposant les coûts d'accès nuls ; et

ii) recherche d'une partition (I_1, I_2) minimisant ces derniers coûts sur t.

Le premier problème étant supposé résolu, on formulera le deuxième en :

$$(I)\quad \begin{cases} \min \sum_{i \in I_2} \xi_i \ \mu(X_i, t) \\ \text{avec} \sum_{i \in I_2} \zeta_i \geq \gamma' \end{cases} \qquad \text{et } I_2 \subset \{1, \ldots n\} \ ; \ I_1 = I - I_2$$

en notant : $\mu(X_i, t) = \sum_{v(v) = x_i} \mu(v)$ et $\gamma' = \sum_{i=1}^{n} \zeta_i - \gamma$

Ce problème a pour dual le problème bien connu du "sac-à-dos" : étant donné n objets, chacun ayant une valeur ($\xi_i \cdot \mu(X_i, t)$) et un volume (ζ_i), et un sac de capacité limitée γ', remplir le sac de façon à emporter une valeur totale maximale :

$$(II)\quad \begin{cases} \max \sum_{i \in I_1} \xi_i \cdot \mu(c_i, t) \\ \text{avec} \sum_{I_1} \zeta_i \leq \gamma \end{cases} \qquad \text{et } I_1 \subset \{1, \ldots n\}$$

Le "sac-à-dos" est un problème NP-Complet (110), mais pas au sens fort, car il admet un algorithme pseudo-polynômial (la Programmation Dynamique), et un schéma d'approximation pleinement polynômial. Ibarra et Kim (102) ont proposé un tel algorithme basé sur les techniques de segmentation du domaine des coûts. De nombreux autres algorithmes à approximation garantie existent (106,136,215).
L'un des plus simples(algorithme glouton)consiste à passer en revue tous les objets dans l'ordre du rapport (valeur/volume) décroissant, tout objet examiné étant mis dans le sac si celui-ci peut le contenir. Moyennant une précaution supplémentaire, cet algorithme en O(nlogn) fournit une solution minorée par 0.5 fois la valeur du maximum exacte ($\varepsilon = 0.5$).

Si l'on cherche une solution exacte, notre problème et son dual sont rigoureusement équivalents, et $\hat{I}_2$ est simplement le complémentaire de la solution optimale $\hat{I}_1$ du "sac-à-dos". Ceci n'est pas vrai relativement à la recherche d'une solution approchée. Comme pour presque tous les problèmes d'optimisation, la borne garantissant le degré d'approximation d'une solution dans la formulation "max" est perdue pour la formulation "min". Plus précisément, elle se transforme en un

majorant absolu non constant : si I_1 est une solution ε - optimal du dual, son complémentaire $I_2 = \{1, \ldots n\} - I_1$ est une solution du primal majorée par :

$$\sum_{I_2} \xi_i \cdot \mu(x_i,t) \leq \sum_{\hat{I}_2} \xi_i \mu(x_i,t) + \frac{\varepsilon}{1-\varepsilon} \sum_{I_1} \xi_i \cdot \mu(x_i,t)$$

(Preuve : en simplifiant les notations, on a :

$\Sigma_1 + \Sigma_2 = \hat{\Sigma}_1 + \hat{\Sigma}_2$; et Σ_1 est ε - optimal :

$\Sigma_1 \geq (1-\varepsilon)\hat{\Sigma}_1$; d'où :

$$\Sigma_2 = \hat{\Sigma}_2 + \hat{\Sigma}_1 - \Sigma_1 \leq \hat{\Sigma}_2 + \frac{\Sigma_1}{1-\varepsilon} - \Sigma_1 = \hat{\Sigma}_2 + \frac{\varepsilon}{1-\varepsilon}\Sigma_1 \qquad \square)$$

Ce majorant non constant n'est pas très satisfaisant. On peut cependant mieux exploiter le schéma d'approximation pleinement polynômial du "sac-à-dos" pour obtenir, pour le problème primal, un algorithme à approximation absolue ou relative garantie. En dehors des cas triviaux où la contrainte est soit partout satisfaite ($\sum_1^n \zeta_i \leq \gamma$), soit ne l'est que pour l'ensemble vide ($\forall i : \zeta_i > \gamma$), on peut procéder ainsi :

- si on désire une approximation à <u>écart absolu</u> du minimum majorée : $\Sigma_2 \leq \hat{\Sigma}_2 + \alpha$; il suffit de résoudre le problème dual avec $\varepsilon = \alpha/(\alpha + \Sigma)$ et $\Sigma = \sum_i \xi_i \cdot \mu(x_i,t)$. La solution obtenue est alors Σ_1 telle que : $(1-\varepsilon)\hat{\Sigma}_1 \leq \Sigma_1 \leq \hat{\Sigma}_1 < \Sigma$, d'où : $\Sigma_2 \leq \hat{\Sigma}_2 + \frac{\varepsilon}{1-\varepsilon}\Sigma_1 = \hat{\Sigma}_2 + \frac{\alpha}{\Sigma}\Sigma_1 < \hat{\Sigma}_2 + \alpha$.

- si on désire une solution à écart relatif majoré : $\Sigma_2 \leq (1+\alpha)\hat{\Sigma}_2$. une procédure en deux étapes pourrait être :

1) recherche d'une solution Σ_1 ε-optimale, ε paramètre initial quelconque : posons alors :

$m_1 = \Sigma_1/(1-\varepsilon)$, d'où : $\hat{\Sigma}_1 \leq m$;

$m_2 = \Sigma - \min\{\xi_i - \mu(x_i,t) \mid 1 \leq i \leq n\}$, d'où : $\hat{\Sigma}_1 \leq m_2$;

$m = \min\{m_1, m_2\}$, d'où : $\hat{\Sigma}_1 \leq m < \Sigma$

2) Recherche d'une deuxième solution Σ_1' ε'-optimale, avec

$\varepsilon' = \alpha / \left[\alpha + m/(\Sigma - m) \right]$. Cette solution vérifie :

$(1 - \varepsilon') \hat{\Sigma}_1 \leq \Sigma_1' \leq \hat{\Sigma}_1 \leq m$, d'où

$$\varepsilon' = \alpha(\Sigma - m) \Big/ \left[\alpha(\Sigma - m) + m \right] \leq \alpha (\Sigma - m) \Big/ \left[\alpha (\Sigma - m) + \Sigma_1' \right]$$

$$\Rightarrow \quad \frac{\varepsilon'}{1-\varepsilon'} \Sigma_1' \leq \alpha (\Sigma - m) \leq \alpha (\Sigma - \hat{\Sigma}_1) = \alpha \hat{\Sigma}_2$$

La solution complémentaire à Σ_1' vérifie alors :

$$\Sigma_2 \leq \hat{\Sigma}_2 + \frac{\varepsilon'}{1-\varepsilon'} \Sigma_1' \leq (1 + \alpha) \hat{\Sigma}_2 \quad .$$

On vérifie que , dans les deux cas, et quelle que soit la valeur de $\alpha > 0$, l'algorithme d'approximation obtenu est polynômial. (Il est donc inexact que le "sac-à-dos" en formulation "min" soit à approximation NP-dur comme on le prétend dans (126)).

La garantie d'approximation qui vient d'être obtenue ne nous est pas très utile dans cette première approche, car en décomposant le problème global en recherche d'un arbre, puis affectation de ses caractéristiques, même une résolution exacte de chacun des sous-problèmes ne permettra pas de borner l'écart à l'optimum global. Par contre, si dans une deuxième approche plus ambitieuse, on cherche à garantir une approximation dans le critère global, alors la résolution du "sac-à-dos min" interviendra dans la construction de l'évaluation heuristique. Cette résolution par un algorithme d'approximation polynômial sera déterminante, non seulement pour la complexité de la recherche, mais aussi pour garantir une bonne heuristique (α - minorante).

Abordons cette approche en notant qu'une résolution globale par un algorithme non déterministe (tel que A1) nécessitera à chaque itération deux choix non déterministes : 1) quelle caractéristique $X_j \in X/u$ mettre dans un noeud u de l'arbre, et 2) si X_j n'a pas encore reçu d'affectation, à quelle mémoire l'affecter ? Un algorithme déterministe aura donc à résoudre une double combinatoire, la deuxième n'étant pas caractérisée uniquement par le seul sommet u en cours de développement.

En effet, le coût d'un connecteur ($\rho_j \mu(u)$ si $j \in I_1$; et $(\rho_j + \xi_j)\mu(u)$ si $j \in I_2$) n'est plus une constante, mais varie en fonction du SL où figure ce connecteur. De même, la place restante en mémoire principale à un moment de la recherche dépend du SL en cours d'exploration. Une heuristique tenant compte des coûts d'accès ne pourra plus être définie comme fonction des sommets u seuls, mais variera également avec le SL auquel appartient u. En fait, l'espace H ne convient plus à la modélisation de la recherche et il faudra revenir à l'arborescence $\mathcal{T}$.

On développera cette arborescence telle que décrit en section (1.5.1.), mais en considérant initialement 2n caractéristiques : chaque $X_j \in X$ est dédoublée en X_j^1 affectée à la mémoire principale et X_j^2 affectée à la mémoire secondaire. Pour traduire le fait qu'une caractéristique ne possède qu'une affectation unique en fin d'exploration, dès que X_j^1 (ou X_j^2) figure dans un noeud d'un arbre partiel τ , on supprimera son double des choix restants. On associe également à τ :
$I(\tau) = \{1, \ldots n\} - \{j \mid X_j^1 \in \tau \text{ ou } X_j^2 \in \tau\}$: indice des caractéristiques ne figurant pas dans τ .

$\gamma(\tau) = \gamma - \sum_{X_j^1 \in \tau} \zeta_j$: place restante en mémoire principale relativement à l'affectation des caractéristiques de τ .

L'heuristique proposée pour guider la recherche est une fonction d'évaluation globale sur les noeuds de l'arborescence $\mathcal{T}$, i.e., sur les arbres partiels. Elle comporte un premier terme, identique à l'heuristique du modèle de base, pour estimer les coûts d'évaluation des caractéristiques, et un deuxième terme estimant les coûts d'accès.

Celui-ci sera basé sur une partition (provisoire) de $I(\tau)$ qui minimise le coût d'accès moyen global des caractéristiques nécessaires pour compléter τ en un arbre complet, tout en respectant la contrainte $\gamma(\tau)$. Pour une caractéristique X_j tel que $j \in I(\tau)$, si X_j est affectée à la mémoire secondaire par la partition précédente, son coût d'accès relativement à la branche pendante u de τ sera :

$$\xi_j \sum_{\substack{v \subset u \\ v(v) = X_j}} \mu(v) \geq \xi_j \, (\mu(u) - q_j(u))$$

La somme de ce coût relativement à l'ensemble des branches pendantes

de τ est minorée par : $\xi_j\ (1 - q_j)$, car $\sum_{w \in \Gamma_i(v)} q_j(w) = q_i(v)$ et sachant que $X_j \notin \tau$, la somme sur toutes les branches pendantes : $\sum q_j(w) = q_j$.

Aussi, on définit la partition provisoire de $I(\tau)$ par la résolution approchée de :

$$\text{(III)} \quad \begin{cases} \min \sum_{i \in I_2(\tau)} \xi_i\ (1-q_i) \\ \text{avec} \sum_{i \in I_1(\tau)} \zeta_i \leq \gamma(\tau) \end{cases} \qquad \text{et} \quad I_1(\tau) = I(\tau) - I_2(\tau)$$

Relativement à cette partition, la fonction d'évaluation heuristique est :

- pour l'arbre partiel vide τ_o :

$$f(\tau_o) = \sum_{i=1}^{n} \rho_i\ (1-q_i) + \sum_{i \in I_2(\tau_o)} \xi_i\ (1-q_i)$$

- pour un arbre partiel τ' obtenu en développant τ selon sa branche pendante u, l'incrément d'estimation est : $f(\tau') = f(\tau) + g(\tau, u, X_j^i)$, tel que :

(i) . Si u est reliée à la caractéristique X_j^1 avec $X_j^1 \in \tau$ <u>ou</u> $j \in I_1(\tau)$:
$g(\tau, u, X_j^1) = \rho_j\ q_j(u)$

(ii) . Si u est reliée à la caractéristique X_j^2 avec $X_j^2 \in \tau$ <u>ou</u> $j \in I_2(\tau)$:
$g(\tau, u, X_j^2) = (\rho_j + \xi_j)\, q_j(u)$

(iii) . Si u est reliée à X_j^1 avec $j \in I_2(\tau)$:
$g(\tau, u, X_j^1) = \rho_j\ q_j(u) - \sum_{i \in I_2(\tau)} \xi_i\ (1-q_i) + \sum_{i \in I_2(\tau')} \xi_i\ (1-q_i)$

(iv) . Si u est reliée à X_j^2 avec $j \in I_1(\tau)$:
$g(\tau, u, X_j^2) = (\rho_j + \xi_j)\, q_j(u) + \xi_j(1-q_j) - \sum_{i \in I_2(\tau)} \xi_i(1-q_i) + \sum_{i \in I_2(\tau')} \xi_i(1-q_i)$

(v) . Si u est reliée à une feuille $g(\tau, u) = 0$.

L'heuristique s'appuie sur une prévision de ce que seront les affectations des caractéristiques dans un arbre. Cette prévision est obtenue par la résolution de (III). Si le développement de τ s'effectue, soit par des caractéristiques

qui figurent déjà dans τ (et qui sont déjà affectées), soit par des caractéristiques affectées selon la partition prévue, alors on poursuit sur la base de la même prévision (cas (i) et (ii)). Si, au contraire, le développement de τ utilise une affectation contraire à celle prévue dans $(I_1(\tau), I_2(\tau))$, alors on corrige l'estimation pour les caractéristiques restantes sur la base d'une nouvelle partition $(I_1(\tau'), I_2(\tau'))$ conforme à l'affectation dans τ' (cas (iii) et (iv)). On peut établir, pour cette heuristique, le résultat suivant :

Proposition 3.9. : Si chaque partition $(I_1(\tau), I_2(\tau)$ utilisée dans le calcul de f est issue d'une résolution α-optimale de (III), alors l'heuristique f est α- minorante et coïncidante.

Preuve :

Coïncidence : on veut établir que pour un arbre complet t :

$$f(t) = \psi(t) = \sum_{j=1}^{n} \rho_j \ \mu(X_j, t) + \sum_{j \in I_2(t)} \xi_j \ \mu(X_j, t) \quad ; \text{ avec}$$

$$\mu(X_i, t) = \sum_{v(v) = X_i} \mu(v) \quad , \text{ et } (I_1(t), I_2(t)) \text{ correspondant à l'affectation dans t.}$$

On ne s'intéressera qu'aux coûts d'accès, car f est linéaire en ρ et μ, et la coïncidence est vérifiée par rapport aux coûts d'évaluation (le terme correspondant dans f est identique à l'heuristique du modèle de base).

- Pour une caractéristique X_j affectée dans t selon la partition initiale $(I_1(\tau_o), I_2(\tau_o))$:

(i) . Si $j \in I_1(\tau_o)$: ξ_j n'intervient ni dans f(t), ni dans $\psi(t)$;

(ii) . Si $j \in I_2(\tau_o)$: ξ_j apparaît dans f(t) avec exactement le même coefficient que ρ_j, et la même chose est vraie pour $\psi(t)$. Comme il y a coïncidence pour ρ_j, il y a aussi coïncidence pour ξ_j.

- Pour X_j dont l'affectation n'est pas conforme à $(I_1(\tau_o), I_2(\tau_o))$:

(iii) . Si $j \in I_2(\tau_o) \cap I_1(t)$: lors du développement du premier noeud de t où figure X_j^1, l'incrément d'évaluation est :

$$g(t, u, X_j^1) = \rho_j \ q_j(u) - \sum_{I_2(\tau)} \xi_i (1-q_i) + \sum_{I_2(\tau')} \xi_i \ (1-q_i)$$

Le terme correctif - $\Sigma_{I_2(\tau)}$ supprime $\xi_j(1 - q_j)$ de f ; et $j \notin I_2(\tau')$: ξ_j n'intervient ni dans f(t), ni dans $\psi(t)$.

(iv) . Si $j \in I_1(\tau_o) \cap I_2(t)$: l'incrément à l'affectation de X_j^2 dans t est :

$$g(t,u,X_j^2) = (\rho_j + \xi_j)\, q_j(u) + \xi_j(1 - q_j) - \Sigma_{I_2(\tau)} + \Sigma_{I_2(\tau')}$$

L'indice j n'intervient ni dans $I_2(\tau)$, ni dans $I_2(\tau')$. Le terme $\xi_j(1 - q_j)$ est pris en compte comme si X_j se trouvait dans $I_2(\tau_o)$, et on se ramène au cas (ii).

(v) . Une caractéristique X_j dont l'indice j change plusieurs fois de classe dans les partitions successives $(I_1(\tau), I_2(\tau))$ au cours du développement de l'arbre ne modifie pas l'évaluation : chaque changement annule le précédent ; seule est prise en compte la position de j au moment de l'affectation de X_j dans t.

(vi) . Finalement, si X_j ne figure pas dans t, alors d'une part ξ_j n'intervient pas dans $\psi(t)$, et d'autre part, $q_j = 1$ car X_j est redondante, et quelle que soit l'affectation de j dans $(I_1(t), I_2(t))$, ξ_j n'intervient pas dans f(t).

<u>α - minoration</u> : On veut établir que $\forall \tau, t \subset \tau$: $f(\tau) \leq (1 + \alpha)\ \psi(t)$.
Vérifions que l'estimation initiale est α-minorante. On sait que pour tout arbre t : $1 - q_j \leq \mu(X_i,t)$, et que :

$$\sum_{j=1}^{n} \rho_j (1-q_j) \leq \sum_{j=1}^{n} \rho_j \cdot \mu(X_i,t)$$

Par ailleurs, si $(I_1(t), I_2(t))$ correspond à la partition des caractétistiques de l'arbre t, alors cette partition est une solution acceptable de (III), et, au mieux, elle en constitue une solution optimale. Un majorant de la solution α-optimale $I_2(\tau_o)$ est donc :

$$\sum_{I_2(\tau_o)} \xi_j (1-q_j) \leq (1+\alpha) \sum_{I_2(t)} \xi_j (1-q_j) \leq (1+\alpha) \sum_{I_2(t)} \xi_j\, \mu(X_i,t)$$

D'où le majorant global de $f(\tau_o)$:

$$f(\tau_o) = \sum_{j=1}^{n} \rho_j (1-q_j) + \Sigma_{I_2(\tau_o)} \xi_j (1-q_j) \leq \sum_{j=1}^{n} \rho_j \cdot \mu(X_i,t) + (1+\alpha) \sum_{I_2(t)} \xi_j\, \mu(X_i,t)$$

$$f(\tau_o) \leq (1+\alpha)\ \psi(t).$$

Un argument similaire peut être repris pour l'estimation d'un arbre partiel :

$1 - q_j + \sum_{(w)=c_j} q_j(w) \leq \mu(X_j, t)$: est vérifié pour tout $t \prec \tau$.

Donc, pour les caractéristiques figurant dans τ et déjà affectées, l'estimation est minorante, aussi bien pour les coûts d'évaluation, que pour les coûts d'accès. □

Pour les autres caractéristiques, l'estimation se base sur une résolution α-optimale de (III), au pire, elle est α-minorante.

Cette fonction d'évaluation privilégie, pour le développement d'un sommet, les caractéristiques qui figurent déjà dans l'arbre et qui sont affectées à la mémoire principale : l'utilisation de toute autre caractéristique X_j entraînerait soit un coût d'accès si X_j est mise en mémoire secondaire, soit une contrainte plus forte sur l'espace $\gamma(\tau)$ restant en mémoire principale et, par là, une valeur du minimum de (III) plus élevée. Dans le cas d'une recherche en profondeur sans backtracking ($\varepsilon = \infty$), l'évaluation conduira pratiquement à prendre pour les noeuds de rang faible (et de probabilité $\mu(v)$ élevée) des caractéristiques en mémoire principale, et cela jusqu'à saturation de la contrainte, puis à affecter à la mémoire secondaire toutes les autres caractéristiques associées aux noeuds de rang élevé ou non utilisées dans l'arbre.

Même si l'heuristique proposée s'avérait très efficace pour guider la recherche, son évaluation serait pénalisante sur les performances de l'algorithme s'il fallait, lors du développement de chaque noeud u, résoudre $|\Phi(u)|$ fois le problème (III) (si aucune des caractéristiques de X/u ne figure déjà dans τ, sur les $2x|\Phi(u)|$ choix possibles, la moitié n'est pas conforme à la partition projetée, ce qui nécessiterait autant de résolutions de (III)). Ce n'est cependant pas le cas grâce à la relation de dominance de la proposition suivante :

Proposition 3.10. : Soit $g_{min}(\tau, u)$ le minimum d'incrément de f pour un développement de (τ, u) conforme à la partition projetée $(I_1(\tau), I_2(\tau))$. Les deux seuls

cas où un développement non conforme à cette partition peut conduire à un gain par rapport à $g_{min}(\tau, u)$ sont :

(i) développement par X_j^1 avec $j \in I_2(\tau)$ et :

$$\rho_j\, q_j(u) < g_{min}(\tau, u) + \alpha \sum_{I_2(\tau)} \xi_i\,(1-q_i) \qquad ; \text{ ou}$$

(ii) développement par X_j^2 avec $j \in I_1(\tau)$ et :

$$(\rho_j + \xi_j)\, q_j(u) < g_{min}(\tau, u) + \alpha \sum_{I_2(\tau)} \xi_i\,(1-q_i)\ .$$

Preuve : Une affectation de X_j non conforme à $(I_1(\tau), I_2(\tau))$ ne conduit à une amélioration de $g(\tau, u, X_j)$ par rapport à l'affectation conforme que si la correction Δ d'estimation des coûts d'accès est positive.

- Pour $j \in I_2(\tau)$: $\Delta = \sum_{I_2(\tau)} \xi_i\,(1-q_i) - \sum_{I_2(\tau')} \xi_i\,(1-q_i)$

  Or la solution $I_2(\tau')$ de (III) sur $I(\tau') = I(\tau) - \{j\}$ est aussi une solution acceptable pour (III) sur $I(\tau)$ (la contrainte $\sum_{I_2(\tau)} \zeta_i$ est la même que celle sur $\sum_{I_2(\tau')} \zeta_i$) ; au mieux, elle en est une solution optimale.
  $I_2(\tau)$ étant α-optimale, si $\hat{I}_2$ est une solution optimale de (III), on a :

  $$\Delta = \sum_{I_2(\tau)} - \sum_{I_2(\tau')} \leq \alpha\, \Sigma^{\wedge}_{I_2(\tau)} \leq \alpha\, \Sigma_{I_2(\tau)}$$

  L'incrément d'estimation est : $g(\tau, u, X_j^1) = \rho_j \cdot q_j(u) - \Delta \geq \rho_j\, q_j(u) - \alpha\, \Sigma_{I_2}$.
  Si $\rho_j\, q_j(u) \geq g_{min}(\tau, u) + \alpha\, \Sigma_{I_2(\tau)}$, alors $g(\tau, u, X_j^1) \geq g_{min}(\tau, u)$.

- Pour $j \in I_1(\tau)$: $\Delta = \Sigma_{I_2(\tau)} - \Sigma_{I_2(\tau') \cup \{j\}}$

  La solution $I_2(\tau') \cup \{j\}$ est acceptable pour (III) sur $I(\tau)$, et comme précédemment:

  $$\Delta \leq \alpha\, \Sigma^{\wedge}_{I_2} \leq \Sigma_{I_2(\tau)} \qquad ; \text{ et}$$

  $$g(\tau, u, X_j^2) = (\rho_j + \xi_j)\, q_j(u) - \Delta \geq (\rho_j + \xi_j)\, q_j(u) - \alpha\, \Sigma_{I_2(\tau)}\ . \qquad \square$$

L'obtention de g_{min} ne requiert pas de calcul supplémentaire par rapport à l'heuristique du modèle de base. La quantité $\alpha\, \Sigma_{I_2(\tau)}$ ne nécessite pas toujours la résolution de (III) sur $I(\tau)$: s'il n'y a pas eu de modification dans les affectations prévues, la partition $(I_1(\tau), I_2(\tau))$ peut être héritée directement du père de τ . Le test des deux inégalités de la proposition est donc très

rapide, et la résolution de (III) n'interviendra que pour les caractéristiques qui les vérifient.

Remarquons que la condition (ii) de cette proposition est très restrictive, car sa vérification implique :

$$\alpha \Sigma_{I_2(\tau)} > p_j q_j(w) - g_{min}(\tau, w) \geq 0 \quad \text{pour } (j \in I_1(\tau) : g_{min} < p_j q_j(w)$$

et $$\xi_j q_j(w) < \alpha \Sigma_{I_2(\tau)} - (p_j q_j(w) - g_{min}(\tau, w))$$

ce qui ne peut pas, en particulier, se produire si f utilise une résolution optimale de (III) ($\alpha = 0$).

Par ailleurs, à la restriction couverte par la condition (i), s'ajoute celle due à la contrainte $\Sigma_{I_1(\tau)} s_i < M(\tau)$: très vite, cette contrainte sera saturée et il ne sera pas possible d'affecter en mémoire principale une caractéristique X_j pour j initialement prévue en $I_2(\tau)$.

Notons, que même si la résolution de (III) se produit beaucoup moins fréquemment que l'évaluation de f, on gagnerait à résoudre (III) par un algorithme qui procède en "différentiel" par des changements locaux sur une solution déjà connue (la partition $(I_1(\tau), I_2(\tau))$ pour obtenir une autre solution à la suite d'une modification du référentiel ($I(\tau') = I(\tau) - \{j\}$)et, éventuellement, de la contrainte. L'algorithme glouton (rapport "valeur/volume" décroissant) est facilement aménageable dans ce sens.

Finalement, rappelons que si le schéma d'approximation utilise une heuristique α-minorante, alors le degré d'approximation de la solution fournie est au plus $[\varepsilon'(1+\alpha) + \alpha]$, ε' étant l'écart relatif à l'estimation du meilleur arbre partiel pendant. Pour garantir une solution ε_0 - optimale, il faudrait résoudre (III) avec $\alpha < \varepsilon_0$, et fournir comme argument d'appel à l'algorithme $\varepsilon = \frac{\varepsilon_0 - \alpha}{1+\alpha}$.
Une précision plus grande dans l'heuristique (α faible) est compensée par une recherche plus sélective (ε grand).

3.3.4. MODELE DES COUTS DE MAINTENANCE

Le coût de maintenance m_i d'une caractéristique X_i intervient sur le coût du processus décisionnel indépendamment de la (ou des) position de X_i dans un arbre t, avec uniquement une pondération 1 ou 0 suivant que X_i figure ou ne figure pas dans t :

$$\psi(t) = \sum_{i=1}^{n} \rho_i * \sum_{v(v)=X_i} \mu(v) + \sum_{X_i \in t} m_i$$

Pour ce modèle également, l'espace de recherche H est insuffisant, car le coût K(u,j) n'est pas intrinsèque au connecteur $\Gamma_j(u)$, mais contient une composante qui dépend du SL où figure ce connecteur. Une modification mineure de l'algorithme AO_ε (plus précisément de la procédure SOUS-LATTICIEL) permettra de prendre en compte cette particularité.

Définissons :

$K(u,j) = \rho_j * \mu(u) + m_j$ si X_j est rencontré pour la première fois dans le SL en cours de développement,

$K(u,j) = \rho_j * \mu(u)$ sinon.

L'estimation heuristique pour la racine de H pourrait être :

$$h(u_o) = \sum_{i=1}^{n} \rho_i * (1 - q_i) + \sum_{q_i \neq 1} m_i \quad ;$$

et l'incrément d'estimation $g_j(u)$, après le développement du sommet par le connecteur $(u, \Gamma_j(u))$ est :

$g_j(u) = \rho_j * q_j(u)$ si $q_j \neq 1$ ou si X_j a déjà été rencontré dans Λ

$g_j(u) = \rho_j * q_j(u) + m_j$ sinon.

Proposition 3.11. : L'heuristique définie par $h(u_o)$ et par la fonction incrément $g_j(u)$ est un estimateur monotone et coïncidant.

Preuve : La monotonie découle immédiatement de : $\forall u, j : g_j(u) \geq 0$.
La coïncidence s'établit par différence relativement au modèle de base. Toute caractéristique X_i tel que $q_i \neq 1$ figure nécessairement dans tout arbre t représentant S. Son coût de maintenance est compté dans l'estimation du coût de la racine : $h(u_o)$. Par ailleurs, g est définie de telle sorte que tout autre terme intervenant dans $\sum_{X_i \in t} m_i$ a été compté une et une seule fois dans l'estimateur. □

Remarque : L'heuristique qui, pour tout j reporterait la prise en compte de m_j au moment où X_j est rencontrée pour la première fois dans Λ, serait également monotone et coïncidante, mais elle serait moins efficace sur un plan algorithmique que celle définie ici, car moins informée.

°0°

3.4. GENERALISATIONS STRUCTURELLES

3.4.1. RESTRICTION DU DOMAINE DES CARACTERISTIQUES ET CONTRAINTES DE SUCCESSION

Une des hypothèses, jusqu'à présent, a été que toute caractéristique dans un système de règles S est une fonction partout définie sur E. Sa conséquence pratique est que les n caractéristiques peuvent être évaluées dans un ordre quelconque, ce qui permet entière liberté dans la génération d'un arbre de décision représentant S. Dans de nombreuses applications (Cf. exemple dans le chapitre 5), cette hypothèse n'est pas valide et doit être remise en cause.

Soit donc, pour i = 1, ..., n, $E_i \subset E$ le domaine de définition de X_i. Une première interprétation pourrait être que si X_i est testée dans un état $e \in (E - E_i)$, une sortie indiquant que X_i est "Non-définie" est obtenue. Cela revient à étendre chaque domaine de définition de E_i à E et à ajouter au domaine d'application de chaque caractéristique X_i une $(z_i + 1)^{ème}$ valeur, laquelle n'apporte qu'une information indirecte et peu exploitable par le processus décisionnel. De plus, cette interprétation exige qu'à chaque évaluation d'une caractéristique X_i, le test très coûteux "$e \in E_i$?" soit exécuté. En fait, on reporte la difficulté en se ramenant au modèle de base.

Une deuxième approche considèrera que le test d'une caractéristique non définie conduit à une erreur et ne doit avoir lieu dans aucun arbre. On restreint alors la définition d'un arbre t représentant un système S au cas où pour tout noeud v de t : $\mathcal{V}(v) = X_i \Rightarrow$ l'ensemble d'états associés au r-cube v est inclus dans E_i. Ceci introduit une contrainte sur la "représentabilité" d'un système S : S est représentable si dans tous les cas, le test "$e \in E_i$?" peut être effectué uniquement grâce à l'évaluation d'autres caractéristiques du système, lesquelles devront donc être définies (il doit, en particulier, exister au moins une caractéristique partout définie sur E).

En adoptant cette approche, on s'intéressera dans ce qui suit, au cas où la restriction du domaine d'une caractéristique X_i quelconque peut être faite

par des règles du type :

"Si ($\mathcal{F}$) alors X_i n'est pas définie" ;

$\mathcal{F}$ étant une ebf portant sur des conditions où ne figure pas X_i.

Une telle règle, appelée contrainte de succession, sera représentée par un ou plusieurs termes de U : u = (u(1), ..., u(i - 1), θ , u(i + 1),...,u(n)) ; le symbole "θ" comme $i^{ème}$ coordonnée de u signifiant que X_i n'est pas définie sur l'ensemble des états e $\in$ E tel que :

$$(X_1(e), \ldots, X_{i-1}(e) \ , X_{i+1}(e), \ldots, X_n(e)) \subset (u(1), \ldots, u(i-1), u(i+1), \ldots u(n)).$$

Remarque : Pour garder à l'espace de représentation U la même sémantique que précédemment, on admettra que toute caractéristique X_i est partout définie sur E, mais qu'on s'interdit de l'évaluer sur (E - E_i) (car, par exemple, son coût d'évaluation y devient infini). La contrainte de succession s'énoncera :
"Si ($\mathcal{F}$) alors X_i n'est pas évaluable".
Les termes correspondant à cette contrainte pourront être manipulés comme tous les termes des autres règles du système (regroupement, recouvrement, développement), en interprétant le symbole "θ" comme un "φ", et en prenant la précaution de ne pas perdre l'information : u(i) = $\theta \Leftrightarrow$ X_i est non évaluable.

Une condition nécessaire de représentabilité d'un système est :

Proposition 3.12. : Soit u une contrainte de succession telle que $\forall i$ u(i) $\neq \varphi$ (i.e., u(i) = θ, ou u(i) est explicite sur $\{0, \ldots, z_i - 1\}$). Si u recouvre plusieurs règles de décision, i.e., si $\exists j$, $k \neq j$ tel que (u $\cap$ R_j) $\not\subset$ D et (u $\cap$ R_k) $\not\subset$ D, alors S n'est pas représentable.

Preuve : (par l'absurde) - Soit t un arbre représentant S. t réalise une partition de ($\overset{n}{\underset{i=1}{X}} z_i$) admissible pour S. Dans cette partition, la contrainte de succession u est recouverte par, au moins, deux feuilles l_1 et l_2 de t. Or, par hypothèse, u ne contient aucune coordonnée "φ". Elle ne peut donc être partitionnée par l_1 et l_2 qu'en développant une coordonnée "θ", i.e., en testant dans t une caractéristique non évaluable. □

On déduit de cette proposition que S n'est représentable que si toute contrainte de succession peut être développée (suivant ses "φ") en termes, chacun inclus dans une règle unique ou dans D. On s'évitera donc de définir un ensemble particulier pour les contraintes de succession, en se ramenant à un système $S = \{X, A, D, R\}$ dans lequel toutes les contraintes sont exprimées dans les différentes règles $R_1, \ldots, R_a$; avec pour $u \in R_j$, $u(i) \in \{0, \ldots, z_i - 1\} \cup \{\varphi, \theta\}$.
On étendra la notation $\Phi(u) = \{i \in \{1, \ldots, n\} \mid u(i) = \varphi \text{ ou } u(i) = \theta\}$.

Remarques :

1) Une contrainte de succession sur un terme de dépendance peut être ignorée (θ confondue avec φ) : il est, en effet, inutile de tenir compte de ce qui n'est pas évaluable dans un état par définition impossible.

2) Ayant considéré chaque X_i comme partout définie sur E, et ayant interprété le symbole "θ" à un niveau purement opérationnel, ce qui a été établi sur la consistance et la complétude d'un système de règle demeure valide avec ou sans contraintes de succession. En particulier, l'algorithme VERIFICATION est directement applicable en considérant une coordonnée "θ" comme équivalente à un "φ".

Pour énoncer une condition nécessaire et suffisante de représentabilité d'un système S, définissons un sous-système associé à un terme u :

$S/u = \{X/u, A/u, D, R/u\}$ par :

$X/u = \{X_i \in X \mid i \in \Phi(u)\}$; et

$\exists j, k \neq j$ tel que $(u/i \leftarrow j) \notin D$ et $(u/i \leftarrow k) \notin D$; et

il n'existe aucune contrainte de succession v tel que $v \subsetneq u$ et $v \cap D \neq \emptyset$ }

A/u et R/u gardent la même définition que dans le modèle de base.

Notons que pour l'hypercube entier : $X/u_o \neq X$ (l'égalité a lieu dans le modèle de base) ; X/u_o est restreint aux seules caractéristiques évaluables sur tout E.

□

Proposition 3.13. : S, un système consistant et complet, est représentable ssi pour tout $u \in U$, l'une des trois conditions suivantes est vérifiée :
1) $u \subset D$; 2) $|R/u| = 1$; ou 3) $X/u \neq \emptyset$.

Preuve (condition suffisante) : l'algorithme non déterministe A1 trouverait à chaque itération, soit une caractéristique évaluable ($X/u * \emptyset$), ce qui permet de poursuivre la génération de l'arbre ; soit une règle de décision unique ($|R/u| = 1$) ou une dépendance, ce qui permet de transformer une branche en feuille ou de l'élaguer.
(Condition nécessaire, par l'absurde) : admettons que u ne vérifie aucune des trois conditions et que t représente S. $X/u = \emptyset$ et $|R/u| > 1$ entraînent que $\Phi(u) \neq \emptyset$ et $\forall i \in \Phi(u)$, il existe $v \subset u$ tel que $v(i) = \theta$. Soit $v_1, \ldots, v_r$, $r \geq 2$ tous ces sous-termes de u développés tels qu'aucun ne contienne de coordonnée "φ". Ou bien, l'un deux v_j recouvre plus d'une règle ($|R/u| > 1$), et suivant la proposition précédente, S n'est pas représentable ; ou bien il existe au moins deux sous-termes v_1 et v_2, chacun inclus dans une règle unique, tels que leur intersection non vide n'est pas dans D ($v_1(i) = \theta$, $v_2(i) \neq \theta$, et $v_1(k) \neq \theta$, $v_2(k) = \theta$, avec $i,k \in \Phi(u)$), et dans ce cas, S est inconsistant. □

Cette proposition ne fournit pas un test opérationnel simple de représentabilité d'un système. Elle est pourtant très importante, car on en déduit facilement que les contraintes de succession n'augmentent en rien la combinatoire du problème de recherche d'un arbre représentant un système, mais, au contraire, diminuent cette combinatoire. En effet, l'algorithme A1 peut faire un choix non déterministe quelconque dans X/u, si ce choix conduit plus tard à un blocage sur une branche (absence de caractéristique évaluable sur un terme non homogène), tout autre choix dans X/u aurait conduit aussi à un blocage, car S n'est de toute façon pas représentable. A1 n'aura jamais à revenir en arrière, et une séquence de choix arbitraires déterminera la représentabilité ou la non représentabilité du système.

Abordons alors le problème de l'optimisation d'un système de règles consistant, complet et représentable. Moyennant la transposition à H de la restriction de X/u précédente, i.e., $\Gamma_j(u)$ est un connecteur de H ssi $X_j \in X/u$, l'espace de recherche du modèle de base demeure valide. Sa distribution de coût également.

Notons que S est représentable ssi H ne contient aucun sommet non terminal sans connecteur. Soit alors :

q_j : probabilité de se trouver dans un état où une décision est accessible sans évaluer X_j, compte-tenu des contraintes de succession ; et

$\overline{q_j}$: même quantité que q_j, en ignorant les contraintes de succession ;

$q_j(u)$ et $\overline{q_j}(u)$ sont les probabilités d'avoir les événements précédents joints à celui de se trouver dans un état de u.

Proposition 3.14. : $\forall u \in U,\ j \in \Phi(u)$: $\overline{q_j}(u) \geq q_j(u)$.

Preuve : La remarque 4 (page 176) demeure valide ici :
$q_j = \sum \mu(\ell)$ sur l'ensemble des feuilles l d'un arbre t_j tel que $X_j \notin \nu(\Gamma^-(l))$; t_j étant un arbre qui, sur tout chemin, n'évalue la caractéristique X_j qu'en dernier ressort, lorsqu'un test reste néanmoins nécessaire et qu'aucune autre caractéristique n'est disponible (i.e. sur une branche telle que $X/u = \{X_j\}$).

Les contraintes de succession imposent à un arbre t_j de tester X_j dans un chemin au moins plus tôt qu'il ne le ferait sans les contraintes : X_j apparaîtra dans davantage de feuilles de t_j, d'où $q_j \leq \overline{q_j}$. □

Proposition 3.15. : l'estimation $h(u) = \sum_{j \in \Phi(u)} \rho_j \left[\mu(u) - q_j(u) \right]$ est une heuristique monotone et coïncidente.

Preuve : la monotonie est immédiate : $g_j(u) = \rho_j * \overline{q_j}(u) \geq \rho_j\, q_j(u) \geq 0$.
Si l'on ignore les contraintes de succession, l'estimation proposée est celle du modèle de base, la structure de coût est la même que dans ce modèle, d'où la coïncidence. □

Remarquons que l'estimation initiale du sommet u_o est plus petite que celle qu'on aurait pû avoir en tenant compte des contraintes de succession : $h(u_o) \leq \sum_{j=1}^{n} \rho_j\ (1 - q_j)$; l'incrément d'estimation $g_j(u)$ est, par contre, plus grand ; mais globalement, l'estimation risque d'être moins efficace avec les $\overline{q_j}$ qu'avec les q_j.

Pour pouvoir améliorer l'heuristique, il faudrait déterminer les q_j, i.e., pour tout terme u d'une règle R_i, il faudrait répondre à la double question : u figure-t-il dans un terme v dont X_j est consensus ? et, existe-t-il un arbre représentant S et admettant v comme feuille ?
Cela revient, en fait, à déterminer un arbre t_j pour chaque caractéristique.

Bien que ce calcul ne doit être fait qu'une seule fois (chaque t_j fournit l'ensemble consensus C_j), il représente un investissement important pour une amélioration incertaine de l'heuristique. C'est une complication d'autant plus inutile que le peu perdu dans l'efficacité de h est compensé par la réduction de la combinatoire du problème due aux contraintes de succession.

Les mêmes considérations peuvent être invoquées sur la question de la redondance des caractéristiques dans un système contraint : les $\overline{q}_j$ ne constituent plus qu'un indicateur nécessaire (mais non suffisant) de redondance, il faudrait non seulement déterminer un arbre t_j, mais tester pour chacune de ces feuilles l tel que $X_j \notin V(\Gamma^{-1}(l))$ que $l \cap D = \emptyset$. On évitera ce calcul coûteux, en espérant que les réductions apportées par les tests de redondance dans le modèle de base soient compensées ici par les contraintes de succession.

3.4.2. REGLE AUTRE

La règle "Autre" permet d'assurer implicitement la complétude d'un système. Elle spécifie l'action (ou les actions) à prendre pour tout état autre que ceux des règles explicites de décision. Remarquons que le test de consistance d'un système est le même avec ou sans règle "Autre", le test de complétude ne peut être utile que pour déterminer si cette règle correspond à un ensemble d'états vide ou non.

Comme on l'a déjà mentionné, expliciter les termes d'une règle "Autre" est équivalent au problème NP-Complet de la réalisation d'une expression logique. On tâchera, donc, de traiter un système avec règle "Autre" sans avoir à expliciter cette règle.

Soit R_A la règle "Autre" et $\mu(R_A)$ la probabilité de se trouver dans un état correspondant à cette règle :

$$\mu(R_A) = 1 - \sum_{v \in S} \mu(v)$$

$$\mu(R_A \cap u) = \mu(u) - \sum_{v \in S} \mu(v \cap u)$$

(par $v \in S$, on ne fait référence qu'aux termes explicites du système).

L'espace de recherche H pour un système muni de la règle "Autre" est le même que dans le modèle de base, en précisant qu 'un sommet u est terminal :

- si $|R/u| = 1$ et $u \subset R_j$, à u correspond alors l'action A_j si $R/u = \{R_j\}$;
- si $R/u = \emptyset$ et $u \subset D$, à u correspond alors l'action de la règle "Autre" ;
- si $u \subset D$: sommet terminal à élaguer.

Soit C_i l'ensemble consensus de la caractéristique X_i, en ne tenant compte que des termes explicites de S, et $q_i = \sum_{v \in C_i} \mu(v)$. Il est simple de voir que q_i est supérieure ou égale à la probabilité de ne pas évaluer X_i dans le PDF, compte-tenu de la règle "Autre". L'heuristique proposée est :

$$h(u) = \sum_{j \in \Phi(u)} \rho_j \left[\sum_{v \in S} \mu(v \cap u) - q_i(u) \right]$$

Proposition 3.16. : h est une estimation monotone et coïncidante.

Preuve : monotonie :

$$K(u,j) + \sum_{v \in \Gamma_j(u)} h(v) = \rho_j \mu(u) + \sum_{i \neq j} \rho_i \left(\sum_{w \in S} \mu(w \cap u) - q_i(u) \right)$$

$$= h(u) + \rho_j \left[\mu(u) - \sum_{w \in S} \mu(w \cap u) + q_j(u) \right]$$

$$K(u,j) + \sum_{v \in \Gamma_j(u)} h(v) = h(u) + \rho_j \left(q_j(u) + \mu(R_A \cap u) \right)$$

L'incrément d'estimation est donc $g_j(u) = \rho_j(q_j(u) + \mu(R_A \cap u)) \geq 0$

Coïncidence :

- si u est un sommet terminal inclus dans une règle unique R_j, alors pour tout $v \in R_j$ tel que $v \cap u \neq 0$, et pour tout $i \in \Phi(u)$, v figure dans un terme de C_i et $\Sigma \mu(u \cap v) = q_i(u)$;

- si u est terminal avec $R/u = \emptyset$, alors $\forall v \in S$: $v \cap u = \emptyset$ et $q_i(u) = 0$;

□

3.4.3. SYSTEMES DE REGLES RECURSIFS INTERCONNECTES

On a considéré, jusqu'à présent, qu'un processus décisionnel s'arrêtait sitôt que la règle de décision correspondante à l'état du système était déterminée. On ne prenait pas en compte dans le PDF la partie action des règles. Mais, dans de nombreuses applications, l'exécution de l'action d'une règle conduit le plus souvent à la poursuite du processus décisionnel. Aussi, est-il intéressant de distinguer deux types de règles :

- celles dont l'action modifie éventuellement l'état du système, puis arrête le processus décisionnel ; et

- celles qui, en plus d'une modification d'état, conduisent à l'appel d'un autre (ou du même) système de règles pour déterminer et exécuter une règle valide sur le nouvel état.

Le processus décisionnel consiste alors en un enchaînement (ou récursion) de PDF élémentaires sur un ou plusieurs sous-systèmes de règles, et cela jusqu'à la rencontre d'une règle d'arrêt.

Notons qu'il est équivalent de considérer toutes les règles comme des opérateurs de changement d'états, et de distinguer certains états particuliers comme terminaux, quitte à définir ces derniers par une ou plusieurs règles.

L'analyse d'un système de règles contenant des appels récursifs sera similaire à celle que font Moret, Thomason et Gonzalez [164] sur les fonctions booléennes récursives. Dans notre cas, l'interprétation d'un appel récursif peut être la suivante : la règle R_j amène le système d'un état e à un état e', pour lequel on cherche la nouvelle règle applicable R' ; comme l'état e' n'est pas directement accessible à partir de R_j et de ce qui a été connu de e (i.e., la feuille l correspondante à e). La seule façon d'obtenir R' est de passer par une nouvelle séquence d'évaluation de caractéristiques.

Soit $\{R_1, \ldots, R_r\}$ les règles dont l'actualisation comporte un appel récursif, et $\{R_{r+1}, \ldots, R_a\}$ les règles d'arrêt. Notons $u_r = \sum_{j=1}^{n} \sum_{v \in R_j} \mu(v)$: probabilité d'appliquer une règle avec appel récursif. On a :

Proposition 3.18. : si t est un arbre représentant S, ε - optimal relativement au critère Ψ (coût moyen d'accès à une règle quelconque indépendamment des récursions), alors t est également ε - optimal relativement au coût moyen d'accès à une règle d'arrêt, et son coût dans ce dernier critère, est :

$$\Psi(t) = \Psi(t) \times 1/(1 - \mu_r).$$

Preuve : Le premier passage dans t coûte $\Psi(t)$. Avec une probabilité μ_r, ce passage conduira à une feuille correspondante à une des règles $\{R_1, \ldots, R_r\}$, entraînant un nouveau passage de coût $\Psi(t)$, ... etc. D'où :

$$\Psi(t) = \Psi(t) \left[1 + \mu_r + \mu_r^2 + \ldots \right] = \Psi(t) \sum_{k=0}^{\infty} \mu_r^k = \Psi(t) \cdot \frac{1}{1-\mu_r} .$$

Comme μ_r ne dépend que de S et pas de t, si t est ε - optimal relativement à Ψ, il l'est aussi relativement à Ψ. □

En pratique, on construira l'arbre t représentant S en négligeant les appels récursifs, puis on ajoutera sur t un arc de retour entre chaque feuille l tel que $\mathcal{L}(l) = A_k$, $k \leq r$, et la racine de l'arbre v_1, et on actualisera le coût de t par $(1 - \mu_r)$.

Cas de plusieurs sous-systèmes de règles s'appelant mutuellement : soient $S_1, \ldots, S_s$ un ensemble de sous-systèmes portant sur le même espace d'état E, mutuellement connectés, et tel que pour tout i, $j \neq i$: toute caractéristique de S_i est logiquement indépendante de toute caractéristique de S_j (donc, en particulier, distincte). Notons μ_{ji} la probabilité totale des règles de S_i contenant un appel à S_j ; μ_{ii} la probabilité d'un appel récursif sur S_i.

Admettons que chaque sous-système soit traité isolément et sans tenir compte de ses récursions propres ; et que pour chaque S_i on ait obtenu ainsi un arbre t_i de coût Ψ_i le représentant. Il est possible, en généralisant le raisonnement précédent, de montrer que si chacun des t_i est ε - optimal, alors la structure totale est ε - optimale et son coût est une fonction linéaire (à coefficients positifs) des Ψ_i.

Exemple 3.5. : Soit la structure suivante (figure 3.5.a) : début du processus sur S_1, puis arrêt, ou appel récursif, ou appel de S_2 ; et, dans ce dernier cas, le processus peut soit se poursuivre sur S_1 ou S_2, soit s'arrêter. On peut remplacer t_2 par un arbre (infini) t'_2 sans récursion et de coût $\Psi'_2 = \Psi_2 \times 1/(1-\mu_{22})$ (schéma figure 3.5.b). Ici, le coût du processus décisionnel sans récursion est $\Psi = \Psi_1 + \mu_{12}\,\Psi'_2$, et la probabilité de retour sur t_1 est $\mu_r = \mu_{11} + \mu_{12}\ \mu_{22}$. On remplace la structure précédente par un arbre unique t de coût Ψ et on se ramène au cas de la proposition 3.13. (figure 3.5.c). D'où, le coût global :

$$\Psi = \frac{\Psi_1 + \mu_{12} \cdot \Psi_2 \cdot 1/(1-\mu_{22})}{1 - (\mu_{11} + \mu_{12} \cdot \mu_{22})}$$

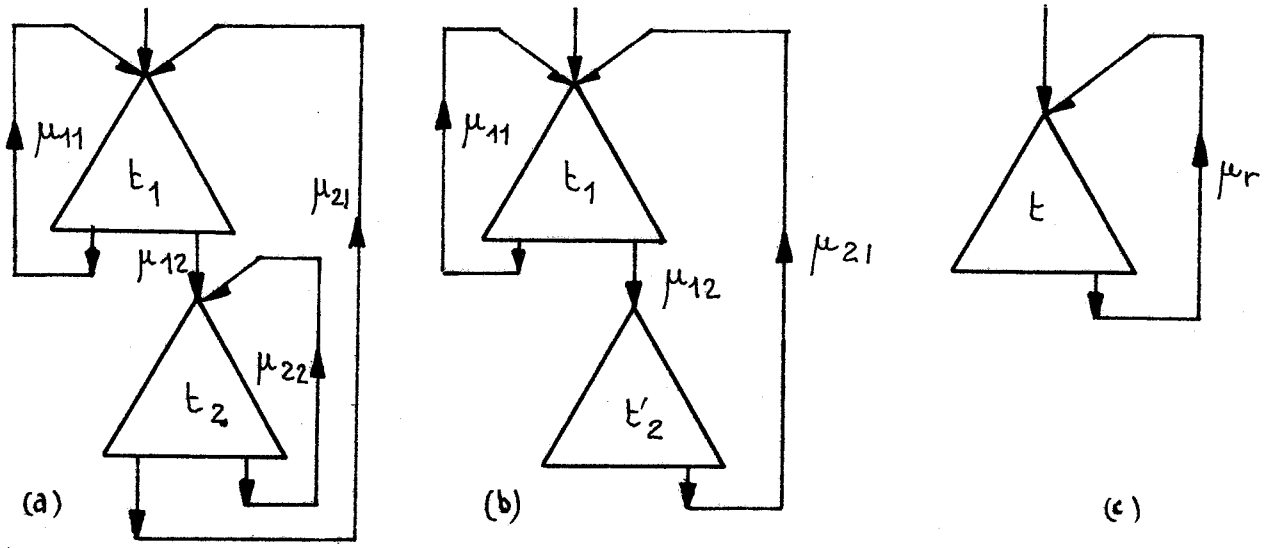


FIGURE 3.5.

Remarques :

1) La structure de l'exemple précédent est symétrique, mais le coût du processus décisionnel démarrant initialement en S_1 n'est pas le même que celui commençant par S_2. La donnée du sous-système initial est fondamentale pour un processus par nature essentiellement séquentiel.

2) Si l'on suppose que toute règle d'un sous-système S_i qui fait appel à un S_j, effectue un changement d'état, l'hypothèse d'indépendance des caractéristiques n'est plus nécessaire. En effet, l'indépendance n'est évidemment pas vérifiée dans le cas d'un appel récursif, cas pour lequel la proposition fut initialement établie (Si la règle R_k fait un appel récursif sur S_i sans changer d'état au préalable, on retraversera l'arbre t_i par le même chemin, aboutissant de nouveau à R_k, ..., et ainsi de suite sur une boucle infinie).Par contre, $R_k \in S_i$ peut faire un appel sans changement d'état à S_j, $j \neq i$, et dans ce cas, l'indépendance des caractéristiques est nécessaire pour justifier que la non prise en compte dans S_j de l'information acquise par le processus décisionnel sur S_i n'affecte pas l'optimalité globale du système. Si R_k avait procédé à un changement d'état (ou était susceptible de le faire), cette information aurait, de toute façon, été perdue (ou non exploitable).

La séquence de réduction de boucles et de transformation de structure illustrée dans l'exemple précédent, devient rapidement très longue et fastidieuse pour un système un tant soit peu complexe. L'analogie entre cette séquence et les techniques de réduction de boucles sur des schémas fonctionnels d'asservissements n'échappera pas au lecteur familier des méthodes de calcul des fonctions de transfert. Une de ces méthodes, s'appuie sur la formule de Masson, donnant le flot entre deux sommets d'un graphe de fluence. Elle est systématique et facilement implémentable. Pour pouvoir l'utiliser, nous avons développé une procédure qui traduit une structure de systèmes récursifs interconnectés en un graphe de fluence, tél que le flot entre deux noeuds conventionnels de ce graphe corresponde au coût du PDF sur la structure.

°0°

3.5. CONCLUSION

Ce chapitre a développé une instance de l'algorithme de recherche heuristique AO_{ε} pour l'optimisation des PDF.

Plusieurs propriétés caractérisant la structure particulière du problème ont été établies, et exploitées pour simplifier certaines phases de l'algorithme et proposer des stratégies efficaces de recherche.

Pour chacun des six modèles de coûts développés au Chapitre 1, une fonction d'évaluation heuristique a été proposée et ses propriétés assurant l' ε-admissibilité de l'algorithme ont été démontrées. On a fait de même pour certaines structures particulières de PDF.

Le domaine d'utilisation de cet algorithme sera précisé au chapitre suivant par le développement d'un modèle empirique de son comportement, et par la caractérisation de la complexité intrinsèque du problème traité, en optimisation exacte ou approchée.

°0°

CHAPITRE 4

COMPLEXITE DE L'OPTIMISATION DES PROCESSUS DECISIONNELS FERMES

4.1. INTRODUCTION

4.2. CARACTERISATION DE LA COMPLEXITE DU PROBLEME D'OPTIMISATION DES PDF

4.2.1. Dimension d'un système de règles

4.2.2. Le problème d'optimisation, exacte ou approchée

4.2.3. Le problème d'existence correspondant

4.3. MODELE EXPERIMENTAL DE LA COMPLEXITE MOYENNE D'AO_ϵ

4.3.1. Implémentation

4.3.2. Expérimentation

4.3.3. Modèle et test de stratégies d'exploration

4.4. APPROCHE POUR UNE OPTIMISATION DYNAMIQUE

4.4.1. Domaine d'application de l'algorithme

4.4.2. Evolutions paramétriques

4.4.3. Evolutions structurales

4.5. CONCLUSION

°0°

4.1. INTRODUCTION

Le but de ce chapitre est d'analyser la complexité du problème d'optimisation des PDF et de modéliser, plus précisément, le comportement moyen du schéma d'approximation.

Rappelons les résultats obtenus jusqu'à présent en matière de complexité. Pour un système de a règles et de n caractéristiques :

- l'algorithme non déterministe A1 est en O(p + q) ; p est le nombre de noeuds et q le nombre de feuilles de l'arbre généré. Dans le pire des cas, A1 est en $O\left(\prod_{i=1}^{n} z_i\right)$; et aucune des heuristiques instanciant A1 en un algorithme approché (fonctions Y_1, Y_2, Y_3 ou Y_4) ne réduit cette borne ;

- la programmation dynamique (A2) est en $O\left(\prod_{i=1}^{n} (1 + z_i)\right)$;

- les procédures par séparation et exploration (A3 ou A4) sont en $O(|\mathcal{T}|)$; avec

$$|\mathcal{T}| \simeq 2\ \Xi(n) = 2n \prod_{k=1}^{n-2} (n-k) \prod_{j=1}^{k} z_{i_j}$$

- l'algorithme AO_ε est en $O\left(\prod_{i=1}^{n} (1 + 2z_i)\right)$.

Non seulement aucun de ces algorithmes n'est polynomial, mais, de plus, on caractérisera (section 4.2.) le problème d'optimisation traité dans une classe de complexité qui exclut théoriquement l'existence d'algorithmes polynomiaux, aussi bien pour l'optimisation exacte que pour l'approximation garantie.

Malgré cela, on argumentera en faveur de l'approche de traduction d'un système de règles en arbre de décision; et de la nécessité de disposer d'un modèle de comportement moyen d'un bon algorithme, et d'une caractérisation aussi précise que possible de son domaine d'application.

C'est ce que développe la troisième section du chapitre pour le schéma d'approximation AO_ε sur la base d'une approche empirique : une implémentation et des conditions d'expérimentation sont décrites, une analyse statistique des données recueillies est effectuée, un modèle est proposé et critiqué.

La dernière partie est très prospective et aborde une question encore largement ouverte : comment suivre la dynamique d'un système de règles, dont les paramètres et la structure évoluent, en optimisant un critère global intégrant à la fois les coûts de recherche et de transformation des PDF, et les coûts de décision sur ces PDF ?

°0°

4.2. CARACTERISATION DE LA COMPLEXITE DU PROBLEME D'OPTIMISATION DES PDF

Dans cette section, on répondra par la négative (conditionnelle à $P \neq NP$) aux questions suivantes :

Existe-t-il pour le problème d'optimisation des PDF

- un algorithme exact polynomial ?
- un schéma d'approximation pleinement polynomial ?
- un schéma d'approximation simplement polynomial ?
- un algorithme polynomial d'approximation absolue ?
- un algorithme polynomial d'approximation relative ? (dans un cas particulier).

On discutera également de la position, relativement à la classe NP, du problème d'existence correspondant à l'optimisation des PDF et de la possibilité de trouver des algorithmes polynomiaux sans garantie d'approximation (cf.(111) ou (63) pour une introduction aux notions de complexité discutées ici).

4.2.1. DIMENSION D'UN SYSTEME DE REGLES

La notion de dimension d'une instance d'un problème π, définie comme étant la longueur d'une chaîne de caractères (dans un alphabet donné) codant cette instance, est fondamentale pour la caractérisation de la complexité de π . En toute rigueur, une telle caractérisation ne peut être que relative au schéma de codage utilisé ; mais en pratique, on s'en affranchit en admettant un "codage raisonnable" qui fournirait un code facilement décodable (toute donnée élémentaire de l'instance doit être accessible en temps polynômial) et concis (de longueur

bornée polynômialement par celle de tout autre code facilement décodable)[1].

Il s'agit donc, dans notre cas, de définir une fonction Dimension telle que tout système de règles S puisse être codé par une chaîne de caractères de longueur bornée polynômialement par Dimension (S). La difficulté réside dans le fait que S n'a pas une représentation unique. Sous la forme normale disjonctive dans laquelle le système est supposé être donné, on peut procéder au développement de chaque terme en 0-cubes, ou au contraire, effectuer des regroupements de termes adjacents d'une même règle ou de D. On peut même effectuer une minimisation du nombre de termes (algorithme du type Quine - McCluskey (151)) ; mais cette minimisation est un problème NP-dur et elle n'est pas nécessaire dans l'optique de l'optimisation des PDF.

On admettra, par contre, l'hypothèse peu contraignante que le nombre de termes de S est borné polynômialement par celui de sa représentation minimale.

Notons que malgré cette hypothèse, le nombre de caractéristiques $n = |X|$ et le nombre de règles $a = |R| = |A|$ ne sont pas des paramètres suffisants pour définir Dimension (S), car même dans la représentation minimale, le nombre de termes peut être exponentiel en n et a (exemple : système de n caractéristiques binaires et de a = 2 règles; R_1 comporte tous les 0-cubes ayant un nombre pair de "0", et R_2 contient tous les 0-cubes ayant un nombre impair de "0" : aucun regroupement n'est possible, et la forme minimale de S possède $m = 2^n$ termes).

Si m est le nombre total de termes de S : $m = |D| + \sum_{i=1}^{a} |R_i|$,
on prendra finalement comme définition :

- s'il existe un polynôme de n majorant m : Dimension (S) = n ;

- sinon : Dimension (S) = m.

---

(1) Garey et Johnson (63, p.21) , tout en avouant ne connaître aucun moyen de formaliser cette notion de "codage raisonnable", proposent quelques règles de ce que pourrait être une méthode standard de codage de données structurées.

4.2.2. LE PROBLEME D'OPTIMISATION, EXACTE OU APPROCHEE

Pour caractériser la complexité du problème d'optimisation des PDF, on utilisera une réduction pseudo-polynômiale du problème du transversal minimal.

Soit G = (V, E) un graphe ; un sous-ensemble de sommets $T \subset V$ est un transversal de G ssi toute arête de G a une de ses extrémités dans T. De nombreux problèmes combinatoires se ramènent à la recherche d'un transversal de cardinalité minimale d'un graphe (ou plus généralement, d'un hypergraphe). Le minimum de cette cardinalité est appelé nombre de transversalité de G et est noté :

$$\tau(G) = \min \left\{ |T| \;\middle|\; T \text{ est un transversal de } G \right\} .$$

Karp (110) a établi le résultat suivant : étant donné G un graphe et k un nombre entier, le problème "G admet-il un transversal de cardinalité inférieure ou égale à k ?" est NP-Complet.

(Preuve par réduction du problème de la clique maximale : si G admet une clique de r sommets, G' son complémentaire admet comme transversal les $|V| - r$ sommets qui ne figurent pas dans la clique. "La clique maximale" est lui-même NP-Complet par réduction du problème de la réalisation d'une expression logique).

On peut alors énoncer :

Proposition 4.1. : L'optimisation des PDF est un problème NP-dur au sens fort.

Preuve : Définissons l'application qui à tout graphe G (V, E), $V = \{v_1, \ldots, v_n\}$ et $|E| = m$, associe le système de règles $S = \{X, A, D, R\}$ particulier suivant :

- X contient n caractéristiques binaires, à chaque sommet v_i de V correspond une caractéristique X_i de X ;

- R comporte (m + 1) règles :

  . à l'arête (v_i, v_j) de E correspond la règle R_k qui possède comme unique terme un 0-cube u_k ayant toutes ses coordonnées à 0 sauf la

$i^{ème}$ et la $j^{ème}$ à 1 :

$u_k(i) = u_k(j) = 1$, et $u_k(l) = 0 \quad \forall l,\ l \neq i$ et $l \neq j$

. la règle R_{m+1} possède un 0-cube unique dont toutes les coordonnées sont à 0

- D contient tous les autres r-cubes de U qui ne figurent pas dans R :

. les n 0-cubes ayant une coordonnée à 1 et les autres à 0 ;

. les $(C_n^2 - m)$ 0-cubes ayant (n - 2) coordonnées à 0 et 2 coordonnées i et j à 1, tel que $(V_i, V_j) \notin E$;

. les C_n^3 (n-3)- cubes ayant 3 coordonnées à 1, et (n - 3) autres à φ .

- La distribution de coût sur X est unitaire ($\forall j$, $\rho_j = 1$) ; et la distribution de probabilité sur U associe la probabilité 1 au 0-cube de la règle R_{m+1}, et 0 à tous les autres termes. Notons que S ainsi défini est consistant et complet, et que son nombre total de termes est polynomial en n.

Exemple 4.1. : Le système associé au graphe G suivant est :

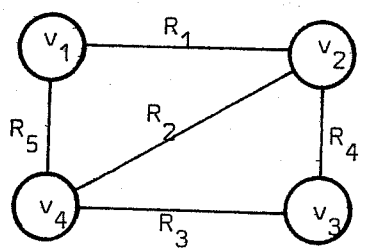


FIGURE 4.1.

$R_1 = \{(1\ 1\ 0\ 0)\}$; $R_2 = \{(0\ 1\ 0\ 1)\}$; $R_3 = \{(0\ 0\ 1\ 1)\}$

$R_4 = \{(0\ 1\ 1\ 0)\}$; $R_5 = \{(1\ 0\ 0\ 1)\}$; $R_6 = \{(0\ 0\ 0\ 0)\}$

$D = \{(1\ 0\ 0\ 0)$; $(0\ 1\ 0\ 0)$; $(0\ 0\ 1\ 0)$; $(0\ 0\ 0\ 1)$;

$(1\ 0\ 1\ 0)$; $(1\ 1\ 1\ \varphi)$; $(1\ 1\ \varphi\ 1)$; $(1\ \varphi\ 1\ 1)$; $(\varphi\ 1\ 1\ 1)\}$.

Chaque règle de S ne comporte comme unique terme qu'un 0-cube, dont tout arbre t représentant S possède exactement (m + 1) feuilles, une par action.

Soit $X_{i_1}, \ldots, X_{i_k}$, les caractéristiques de X rencontrées sur le chemin unique menant à l'action A_{m+1}. Le terme u correspondant à ce chemin est alors défini par : $u(i_1) = u(i_2) = \ldots = u(i_k) = 0$ et $u(j) = \varphi$ pour $j \notin \{i_1, \ldots i_k\}$. Sachant que $R/u = \{R_{m+1}\}$, ce terme u ne recouvre aucun des 0-cubes correspondant aux arêtes de G. Or, ceci ne peut avoir lieu que si, pour chaque arête $(v_i, v_j) \in E$, ou bien $i \in \{i_1, \ldots, i_k\}$, ou bien $j \in \{i_1, \ldots, i_k\}$: l'ensemble $\{v_{i_1}, \ldots v_{i_k}\}$ est donc un transversal de G.

Etant donné la distribution de probabilité sur U, la seule feuille de l'arbre t à avoir un poids non nul est celle correspondante à l'action A_{m+1} ; elle seule intervient dans le coût de t. L'arbre est donc optimal ssi le chemin de sa racine à A_{m+1} est minimal en nombre de noeuds, i.e., ssi le transversal correspondant de G est minimal.

On terminera la preuve en notant que :

- S est construit à partir de G par un algorithme polynômial en la taille de G (au plus en $O(n^4)$), la dimension de S et tous les paramètres qui y interviennent sont bornés polynomialement par n : la transformation de G à S est donc pseudo-polynomiale ;

- le problème du transversal minimal ne faisant pas intervenir de paramètre numérique non borné polynomialement par n, est NP-Complet au sens fort. □

On peut alors affirmer que :

le problème d'optimisation des PDF n'admet pas d'algorithme exact polynômial (à moins que P = NP).

De plus, le qualificatif "sens fort" permettra d'exploiter le théorème suivant (Garey et Johnson (62)) : si π, un problème d'optimisation NP-dur au sens fort, est à paramètres et à critères sur $\mathbb{N}$, et si pour toute instance I de π, la valeur de l'optimum sur I est bornée polynômialement par la dimension de I et par la valeur du plus grand paramètre y intervenant, alors π n'admet pas de schéma d'approximation pleinement polynomial (à moins que P = NP).

Il est relativement simple, dans notre cas, de se ramener aux hypothèses du théorème précédent. Partant d'un système avec des distributions quelconques ρ et μ sur $\mathbb{R}^+$, on obtiendra les distributions sur $\mathbb{N}$ ρ' et μ' en multipliant chaque paramètre par un nombre K assez grand (on pourra prendre $K = 10^{\alpha} . \prod_{i=1}^{n} z_i$, si ρ et μ sont données en représentation décimale avec α digits fractionnaires, le facteur $\prod_{i=1}^{n} z_i$ garantissant que même dans l'hypothèse d'équiprobabilité des sous-termes d'un terme, tout élément de U aura un poids entier). Le critère optimisé étant :

$$\Psi(t) = \sum_{i \leq 1 \leq n} \rho'_i \sum_{\vartheta(v)=X_i} \mu'(v) \quad ;$$

la valeur de l'optimum est entière, et elle est majorée polynomialement (même si l'ensemble $\{v \in V \mid \vartheta(v) = X_i\}$ est exponentiel en la dimension de l'instance, la deuxième somme est majorée par K^2). De plus, tout arbre ε - optimal sur les distributions ρ' et μ' l'est également sur les distributions ρ et μ (or, son coût est $\Psi(t) = 1/K^2 . \Psi'(t)$).

D'où :

<u>le problème d'optimisation des PDF n'admet pas de schéma d'approximation pleinement polynomial (à moins que P = NP)</u>.

Un corrolaire très simple de la proposition 4.1. est :

<u>Proposition 4.2.</u> : Pour tout ε tel que $0 < \varepsilon < 1/n$, le problème de l' ε-optimisation des PDF comportant n ou plus de n caractéristiques, est NP-dur.

<u>Preuve</u> : On reprend la même réduction du problème du transversal minimal d'un graphe que précédemment. Pour le système S obtenu, si t est un arbre de coût K, tout arbre t' strictement meilleur que t est de coût inférieur ou égal à K - 1 :
$(\Psi(t) - \Psi(t') / \Psi(t') > 1/K-1 > 1/n$.

Donc, pour $1/n \geqslant \varepsilon$, tout arbre t ε-optimal est également optimal, et un algorithme d' ε-optimisation des PDF résoud de façon exacte le problème du transversal minimal. □

On en déduit que :

<u>le problème d'optimisation des PDF n'admet pas de schéma d'approximation polynomial (à moins que P = NP).</u>

Finalement, montrons (par un argument similaire à celui de (63) sur le problème du sac-à-dos) qu'une approximation à écart absolu garanti revient à une résolution exacte du problème. Soit A un algorithme qui fournit un arbre dont l'écart <u>absolu</u> à l'optimum est majoré par $\alpha \geq 0$. En partant d'un système S ayant les distributions ρ et μ, passons par un facteur multiplicatif K aux distributions entières ρ' et μ', puis aux distributions $\rho'' = (1 + \alpha)\rho'$ et $\mu'' = \mu'$. L'arbre t obtenu par l'algorithme A a pour coût $\Psi''(t)$ tel que : $\Psi''(t) - \Psi''(\hat{t}) \leq \alpha$; $\hat{t}$ étant une solution optimale (dans les trois distributions). En revenant aux coûts Ψ' : $(1 + \alpha)(\Psi'(t) - \Psi'(\hat{t}) \leq \alpha$.

Or, $\Psi'(t)$ et $\Psi'(\hat{t})$ sont des entiers $\Rightarrow \Psi'(t) = \Psi'(\hat{t})$ et t est également optimal.

D'où le résultat :

<u>le problème d'optimisation des PDF n'admet pas d'algorithme à approximation absolue polynomial (à moins que P = NP).</u>

Une seule alternative reste ouverte : celle de l'éventuelle existence d'un algorithme polynomial d'approximation à écart relatif garanti par une constante. Deux voies sont possibles pour essayer d'exclure cette éventualité :

1) trouver une réduction d'un problème NP-dur au problème de l'approximation des PDF ;

2) trouver un problème dont l'approximation est NP-dur et qui se réduit fortement à l'optimisation des PDF.

La réduction proposée dans la preuve de (4.1.) est une réduction forte (conserve les garanties d'approximation) ; cependant, le problème du transversal minimal d'un graphe n'est pas à approximation NP-dur (si E' est un couplage maximal d'un graphe G = (V, E), alors V', l'ensemble des sommets intervenant dans les arêtes de E', est un transversal de G avec $|V'| \leq 2 \times \tau(G)$; la détermination polynomiale

de E' fournit donc une ($\varepsilon = 1$) - approximation). En fait, relativement peu de problèmes d'optimisation sont à approximation NP-dur pour tout $\varepsilon > 0$ (les plus communs étant, par exemple, le problème du "voyageur de commerce", la programmation linéaire en nombres entiers, la K-partition minimale d'un graphe, ... Cf. (216)) ; et leur structure est assez éloignée de celle des PDF pour qu'une réduction forte polynomiale apparaisse intuitivement. Un problème particulier intéressant est celui du nombre chromatique minimal d'un graphe (61) : son ε - approximation n'est NP-dur que pour $\varepsilon < 1$. Mais, nous n'avons pû trouver qu'une réduction (laborieuse) du problème d'existence "G est-il k-chromatique ?" à celui de l'optimisation exacte (ou à ε près, pour $\varepsilon \leq 1/n$) des PDF.

Un autre problème dont une réduction aurait pû être intéressante est celui de l'ensemble stable maximum d'un graphe ($S \subset V$ est un stable de $G=(V,E)$ ssi deux sommets quelconques de S sont non adjacents). Garey et Johnson ont montré (63) que, soit ce problème est à approximation NP-dur, soit il admet un schéma d'approximation polynomial (excluant l'alternative d'un algorithme à approximation garantie). De plus, il est simple d'établir que si S est un stable maximal de G, alors (V - S) en est un transversal minimal. La transformation de la proposition (4.1.) est donc facilement généralisable, mais uniquement en tant que réduction faible, car malheureusement un problème de maximisation n'est pas fortement réductible à un problème de minimisation (les réductions de ce type dans le théorème 9.10 de (77) sont inexactes).

L'autre alternative évoquée a été explorée pour un certain nombre de problèmes, mais il n'a pas été possible de trouver une réduction qui s'affranchisse d'une contrainte entre le degré d'approximation et la taille de l'instance (du type $\varepsilon \leq 1/n$), et cela à l'exception du cas particulier suivant :

Proposition 4.3. : L'approximation des PDF avec règle Autre est un problème NP-dur.

Preuve : Par réduction du problème de la réalisation d'une expression logique. Soit E une ebf de m clauses sur n variables booléennes $\{y_1, \ldots, y_n\}$ sour forme normale conjonctive. On construit à partir de E le système de règles $S=\{X,A,D,R\}$:

- X comporte (n + m) caractéristiques binaires ;

- R possède m règles explicites en plus de la règle Autre ; la règle R_i correspond à la $i^{ème}$ clause de E et comporte un terme unique u_i, avec :

  . pour $1 \leq j \leq n$: $u_i(j) = 1$ si y_j figure dans cette clause sous forme complémentée, $u_i(j) = 0$ si y_j figure sous forme positive, et $u_i(j) = \varphi$ si y_j n'y figure pas.

  . pour $n < j \leq m+n$, $j \neq n + i$: $u_i(j) = 0$

  . et $u(n + i) = 1$

- D comporte m^2 termes ; à la $i^{ème}$ clause de E correspondent m termes dont les n premières coordonnées sont identiques à celles de u_i, et les m dernières coordonnées choisies de telle sorte que l'union de ces m termes de D soit le complémentaire de u_i dans le m-cube $(u_i(1), \ldots, u_i(n), \varphi, \varphi, \ldots \varphi)$.

- les coûts des (n + m) caractéristiques sont : $p_1 = p_2 = \ldots = p_n = k$; $p_{n+1} = \ldots = p_{m+n} = 1$, et la distribution de poids (entiers) sur U est uniforme.

Exemple 4.2. : Si $E = (\bar{y}_2 \vee \bar{y}_3) \wedge (\bar{y}_2 \vee y_3) \wedge (y_1 \vee y_2) \wedge (\bar{y}_1 \vee y_2)$, on a :

$R_1 = \{(\varphi\ 1\ 1\ 1\ 0\ 0\ 0)\}$

$R_2 = \{(\varphi\ 1\ 0\ 0\ 1\ 0\ 0)\}$

$R_3 = \{(0\ 0\ \varphi\ 0\ 0\ 1\ 0)\}$

$R_4 = \{(1\ 0\ \varphi\ 0\ 0\ 0\ 1)\}$

$D = \{(\varphi\ 1\ 1\ 0\ \varphi\ \varphi\ \varphi), (\varphi\ 1\ 1\ 1\ 1\ \varphi\ \varphi), (\varphi\ 1\ 1\ 1\ 0\ 1\ \varphi),$
$(\varphi\ 1\ 1\ 1\ 0\ 0\ 1), (\varphi\ 1\ 0\ \varphi\ 0\ \varphi\ \varphi), (\varphi\ 1\ 0\ 1\ 1\ \varphi\ \varphi),$
$(\varphi\ 1\ 0\ 0\ 1\ 1\ \varphi), (\varphi\ 1\ 0\ 0\ 1\ 0\ 1), \ldots \}$

La transformation de E à S est polynomiale en n et m ; S est consistant et complet. De plus :

- Si E n'est pas réalisable, alors la négation de E est une tautologie et S est complet sans la règle Autre. On peut alors avoir un arbre représentant S, ne con-

tenant la règle Autre dans aucune de ses feuilles, et utilisant uniquement les caractéristiques X_{n+1}, X_{n+2}, ..., X_{m+n}. Pour k assez grand, l'arbre minimal représentant S aura la structure suivante (figure 4.2.) :

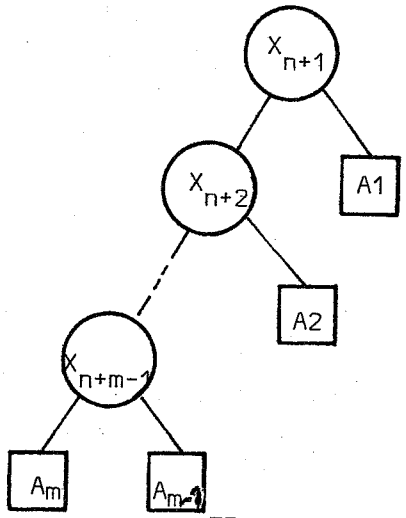


FIGURE 4.2.

son coût est $\psi(\hat{t}) = 1 + 2 + \ldots + (m - 1) + (m - 1) = 1/2\ (m - 1)\ (m + 2)$.

- Si E est réalisable, sa négation n'est pas une tautologie, et la règle Autre n'est pas vide (correspond à tous les termes vérifiant E). Tout arbre représentant S aura au moins une feuille correspondante à cette règle, et au moins une des caractéristiques de $\{X_1, \ldots, X_u\}$ sera nécessaire pour y accéder. Donc : $\psi(\hat{t}) > k$.

Admettons qu'il existe un algorithme A d'approximation des PDF garantissant une solution ε - optimale (ε constante propre à A). Partant d'une ebf E, on construit le système S en prenant $k > (1 + \varepsilon)\ (m + 2)\ (m - 1)/2$, et on applique à S l'algorithme A :

- si l'arbre t obtenu possède plusieurs feuilles correspondant à la règle Autre, alors $\psi(t) > k > (1 + \varepsilon)\ (m + 2)\ (m - 1)/2$ et E est nécessairement réalisable (car, sinon : $\psi(\hat{t}) = (m + 2)\ (m - 1)/2$ et t ne pourrait être ε - optimal) ;

- si, au contraire, t ne comporte pas de règle Autre, alors E n'est évidemment pas réalisable.

Un algorithme polynomial à approximation garantie (quel que soit son degré d'approximation) pour PDF avec règle Autre résoudrait donc polynomialement le problème de la réalisation de E. □

Remarque : Un algorithme polynomial générant un arbre représentant le système précédent pourrait être : tester la séquence X_{n+1}, ..., X_{m+n-1}; et sur chacune des branches droites de X_{n+i}, séparer la règle R_i de la règle Autre par le test d'une caractéristique X_j telle que la variable y_j figure dans la $i^{ème}$ clause de E. L'arbre correspondant à l'exemple (4.2.) possède plusieurs feuilles "Autre" (figure 4.3.) pourtant E n'est pas réalisable . L'hypothèse d'une garantie sur le degré d'approximation, aussi faible soit-elle, est donc nécessaire pour établir la réduction.

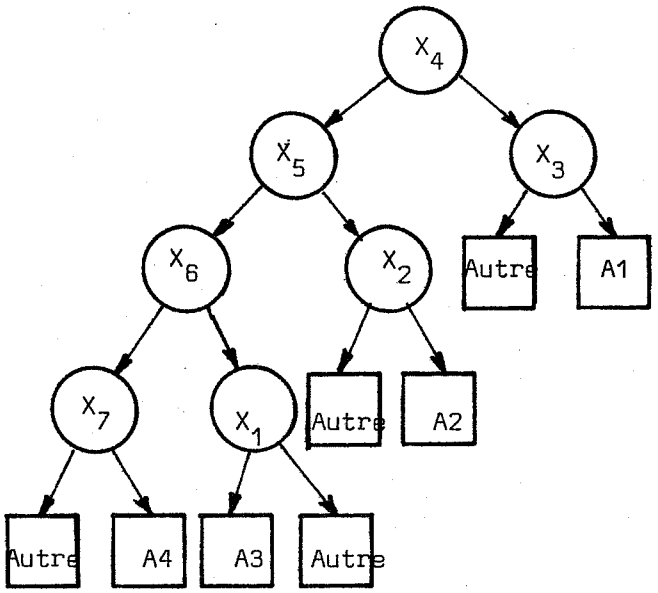


FIGURE 4.3.

Terminons cette section en signalant deux autres preuves dans la littérature sur la complexité de l'optimisation des arbres de décision : Hyafil et Rivest (97) , et Loveland (143) (la première référence est antérieure à la preuve de 4.1., mais elle n'est parvenue, à ma connaissance, que postérieurement). Ces deux résultats portent sur les questionnaires binaires et établissent que le problème de l'existence d'un arbre d'identification de coût inférieur ou égal à une constante donnée, est NP-complet. La preuve de (97) utilise une réduction du problème de la partition d'un ensemble en classes de trois éléments. Celle de

(143) reprend ce même type de réduction sur le problème du recouvrement d'un ensemble par un nombre minimal de ses sous-ensembles pris dans une famille donnée. Un questionnaire étant un PDF particulier, on hérite de sa complexité, mais on ne conserve pas l'appartenance à NP pour les raisons qui vont suivre. Les deux références citées ne se sont pas intéressées à la caractérisation du problème d'approximation.

4.2.3. LE PROBLEME D'EXISTENCE CORRESPONDANT

Jusqu'à présent, nous n'avons pas abordé une question importante sur la complexité de notre problème : est-il dans la classe NP ? ou plus rigoureusement, le problème d'existence correspondant appartient-il à NP ? Ce problème peut s'énoncer ainsi : ayant un système de règles S, les distributions ρ et μ, et une constante K, existe-t-il un arbre de décision t représentant S de coût $\Psi(t) \leq K$?

Par définition, un problème est dans NP ssi il admet un algorithme non déterministe polynomial. Le problème précédent en admettra un ssi pour tout S, ρ et μ, il existe un arbre de décision optimal représentant S qui possède un nombre de noeuds et de feuilles majorés polynomialement par la dimension de S.

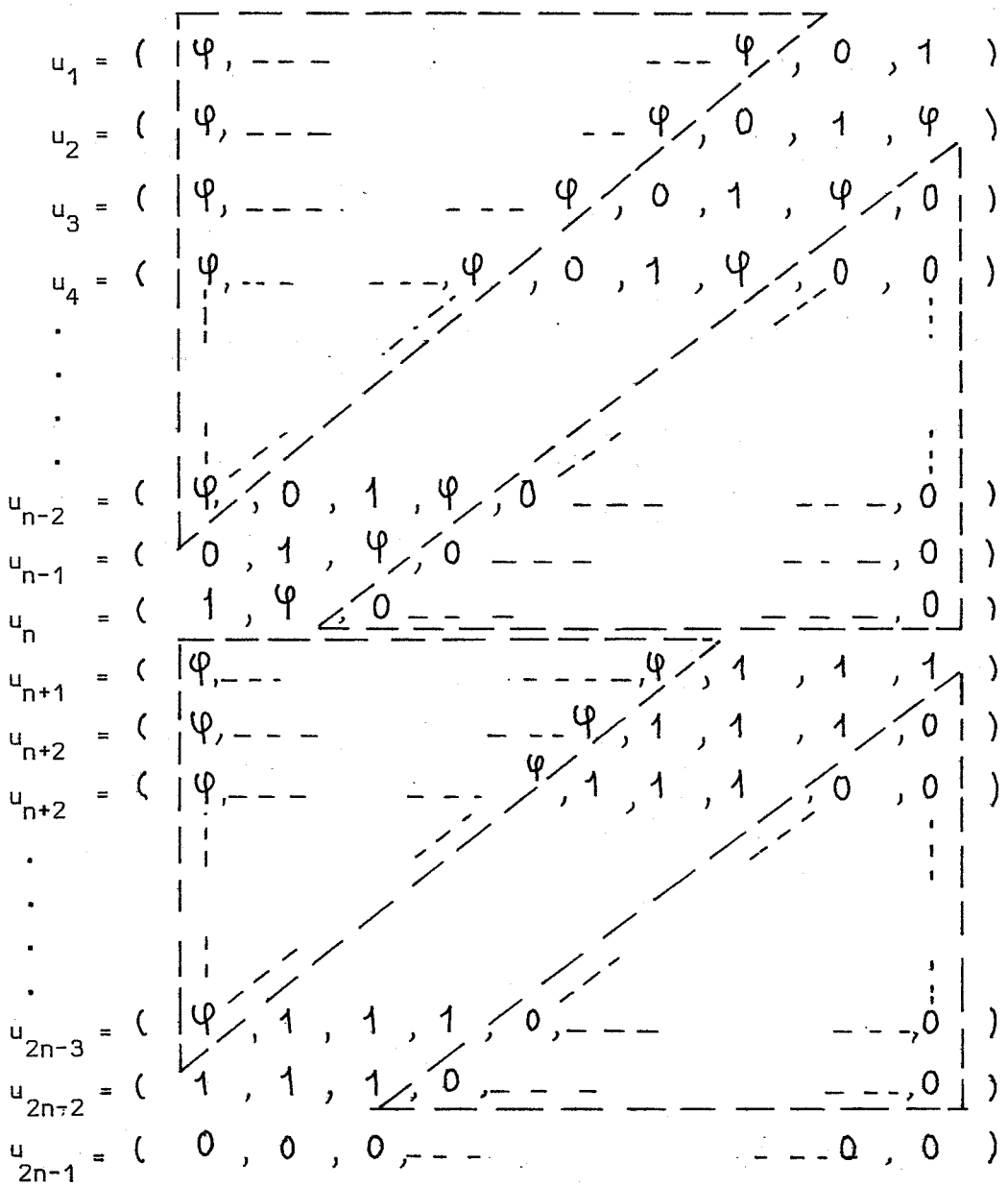
(Preuve très simple : un algorithme non déterministe est celui qui à un noeud de recherche arborescente effectue toujours le bon choix en poursuivant sur la branche menant à une solution si elle existe - ou bien c'est un algorithme disposant d'une machine à parallélisme infini qui se dédouble en autant de copies d'elle-même qu'il y a de branches, chaque copie poursuivant sa propre exploration. La complexité d'un algorithme non déterministe est alors égale à la longueur du chemin le plus court entre le sommet de l'arborescence et une solution. Dans notre cas - A1 explorant τ - cette longueur correspond à (p + q) si l'arbre fourni en solution a p= |X| noeuds et q = |L| feuilles.

On retrouve bien sur O(p + q) comme complexité d'une machine de Turing non déterministe : ayant deviné, sans coût de recherche, un arbre t, celle-ci se contentera de vérifier si $\Psi(t) \leq K$. Or, le calcul de $\Psi(t)$ nécessite une itération par noeud ou feuille de t.

Notons que l'optimalité de l'arbre est nécessaire dans la proposition, car s'il existe $\hat{t}$ de dimension polynômiale, il suffit à un algorithme non déterministe de deviner $\hat{t}$ et de résoudre le problème pour tout K en vérifiant si $\psi(\hat{t}) \leqslant K$; si, au contraire, aucun arbre optimal n'est de dimension polynômiale, alors il suffit de prendre $K = \psi(\hat{t})$ pour exclure une résolution non déterministe polynômiale). □

On peut alors répondre à la question initiale par la négative grâce au contre-exemple suivant :

Exemple 4.3. : Soit le système $S = \{X, A, D, R\}$ de n caractéristiques binaires comportant les (2n-1) termes suivants :



Ces (2n-1) termes sont répartis entre n règles et une relation de dépendance de la façon suivante :

$R_1 = \{u_1, u_2\}$, $R_2 = \{u_3\}$, $R_3 = \{u_4\}$, ..., $R_{n-1} = \{u_n\}$, $R_n = \{u_{n+2}, \ldots, u_{2n-1}\}$.

$D = \{u_{n+1}\}$

On vérifie facilement que tous les termes de S sont 2 à 2 disjoints et qu'ils couvrent tout le n-cube : S est consistant et complet.

On peut également voir que pour tout r-cube u tel que u(n-1)=u(n) = φ , u recouvre plus d'une règle de S : $|R/u| > 1$. Un arbre t qui systématiquement testerait en dernier lieu C_{n-1} ou C_n aurait, dans tous ses chemins, au moins (n-1) caractéristiques ; toutes ses feuilles seront de rang supérieur à (n-2), il possèdera donc au moins 2^{n-1} feuilles et $2^{n-1} - 1$ noeuds.

Soit T la classe des arbres représentant S, qui testent dans tout chemin les caractéristiques $X_1, \ldots, X_{n-2}$ dans un ordre quelconque, puis qui, soit évaluent X_n s'il ne reste que deux actions séparables par cette caractéristique, soit évaluent X_{n-1} et, si nécessaire, X_n (un exemple d'arbre de T pour n = 5 est donné figure 4.4). Prenons une distribution de coût telle que $\rho_1 = \rho_2 = \ldots = \rho_{n-2} = 0$ et $\rho_n > \rho_{n-1} > 0$ et admettons une distribution μ uniforme sur U. On vérifie alors que :

- tout arbre $t \in T$ possède $q = 2^{n-1} + 2^{n-3} + 2^{n-5}$ feuilles, (q-1) noeuds, et son coût est $\psi(t) = (\rho_{n-1} + \rho_n) \times (2^{n-1}+2^{n-4}+2^{n-5})/(2^n-2^{n-3}) = (\rho_{n-1} + \rho_n) \times 19/28$;

- tout arbre $t' \in T$ est de coût $\psi(t') \geq \psi(t) + \rho_{n-1}/(2^{n-1} - 2^{n-3})$.

Donc, pour S, ρ et μ tout arbre optimal appartient à T et possède une taille exponentielle en la dimension de S. □

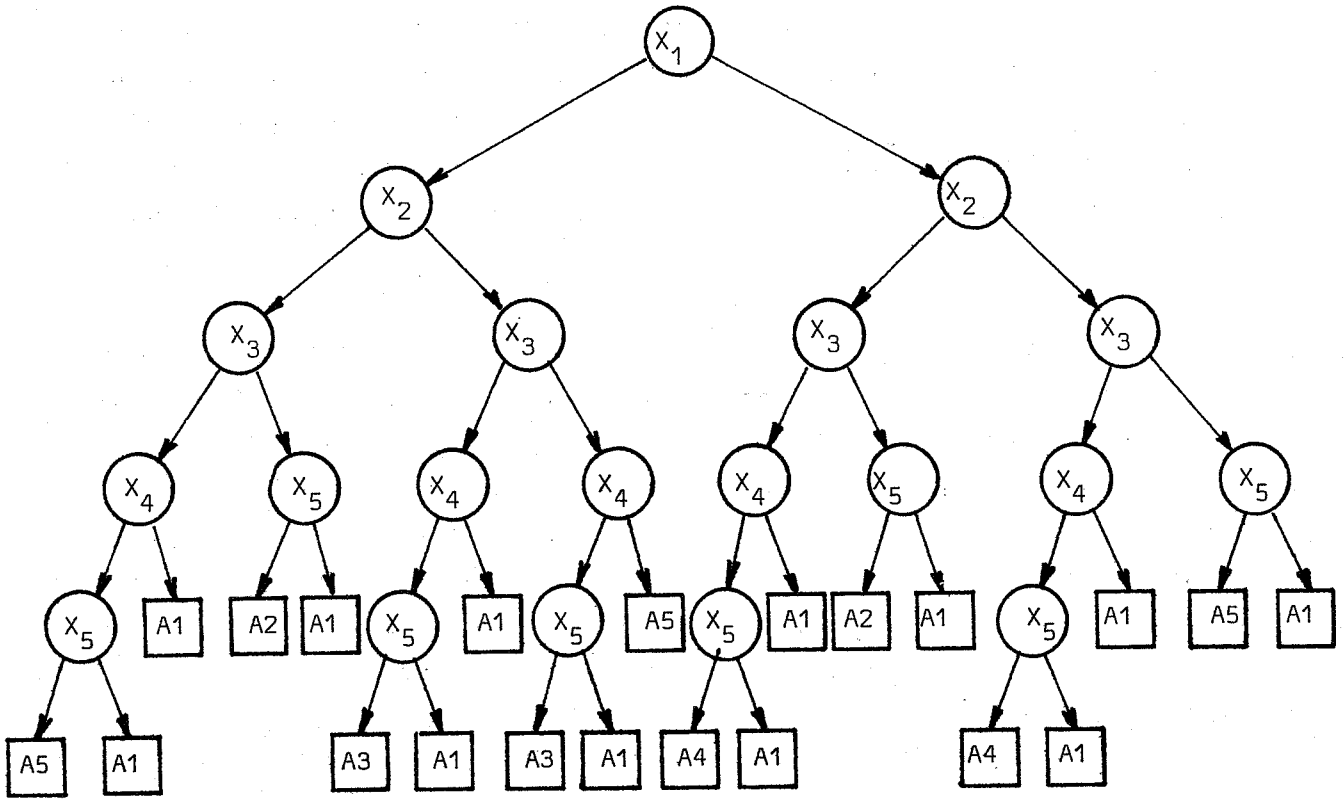


FIGURE 4.4

Ce résultat négatif nécessite quelques commentaires :

1) Si un problème n'admet même pas un algorithme non déterministe polynômial, il admettra encore moins une résolution déterministe polynômiale, exacte ou approchée (de quelque façon que ce soit). Aussi, les caractérisations de la section précédente pourraient sembler inutiles. Mais, en fait, le problème d'existence considéré n'appartient pas à NP, non pas à cause de sa combinatoire propre, mais parce que la quantité de données (un arbre de décision) à vérifier par un algorithme non déterministe n'est pas bornée polynômialement par la dimension des données d'entrée ;

2) Cette composante de la complexité du problème est assez gênante. On peut, cependant, l'éliminer en se restreignant à la classe de systèmes de règles admettant des arbres de décision bornés polynômialement (dont un exemple particulier

sont les procédures d'identification ou questionnaires qui ont exactement autant de feuilles que de règles). On vérifie, sans difficultés, que toutes les réductions de la section précédente s'appliquent à cette classe de systèmes. L'optimisation des PDF est donc un problème complexe (NP-dur au sens fort, à ε - approximation NP-dur, ...), du fait de la combinatoire propre de la recherche et indépendamment de la taille des données à générer, laquelle ne peut qu'augmenter une complexité déjà non polynômiale ;

3) On évite en général, dans la littérature, l'interférence de la dimension des données de sortie sur la complexité d'un problème. (Pour [63] , un problème pouvant générer une quantité de données exponentielle en la dimension des données d'entrée, est mal défini, car on ne peut exploiter raisonnablement toutes les sorties demandées. Si l'affirmation est vraie dans l'exemple cité - Recherche de tous les circuits Hamiltoniens d'un graphe -, elle est inexacte ici : le processus décisionnel ne parcourt, à chaque fois, qu'un chemin de l'arbre, il est donc toujours de complexité temporelle polynômiale, mais peut être de complexité spatiale exponentielle). La complexité est alors analysée soit en fonction de la dimension maximale des données de sortie, soit en redéfinissant la fonction "Dimension" des données d'entrée de façon à avoir des sorties bornées polynômialement. Ici, les deux alternatives reviennent à caractériser la complexité en fonction de $N = \prod_{i=1}^{n} z_i$. Mais, alors, tous les algorithmes présentés jusqu'ici seraient polynômiaux (la Programmation Dynamique est en $\prod_{i=1}^{n} (1 + z_i) \leq N^{\log 3/\log 2}$; AO_ε est en $\prod_{i=1}^{n} (1 + 2\, z_i) \leq N^{\log 5/\log 2}$) et on perdrait l'information, en pratique la plus importante, que la recherche reste exponentielle même lorsque l'arbre trouvé est de dimension polynômiale.

La non appartenance du problème à NP n'exclut pas sa résolution polynômiale par un algorithme approché sans garantie d'approximation (pour l'exclure, il aurait fallu trouver un système S tel que tout arbre le représentant, quel que soit son coût, soit exponentiel). Mais, à ma connaissance, aucun des algorithmes approchés proposés dans la littérature n'a été prouvé polynômial. De plus, on vérifie facilement que le contre-exemple précédent a raison des 4 fonctions Y_1, Y_2, Y_3 et Y_4 discutées en (1.5.2.) : elles conduisent toutes à un arbre de T. En fait, il faudrait se restreindre à un choix heuristique très pauvre qui ne tienne compte ni des coûts, ni des probabilités pour être en mesure, éventuellement, de générer un arbre

polynômial s'il en existe (sur l'exemple 4.3., c'est le cas de l'heuristique du nombre minimal de " Ψ " (39, 196) : on teste d'abord X_n et X_{n-1}, ce qui conduit à un arbre polynômial).

°0°

4.3. MODELE EXPERIMENTAL DE LA COMPLEXITE MOYENNE D'AOε

Les résultats obtenus jusqu'à présent sur la complexité de l'optimisation des PDF sont peu encourageants. Ils pourraient conduire à abandonner entièrement l'approche de compilation d'un système de règles en arbres de décision, au profit d'une interprétation séquentielle de l'ensemble des règles dans un certain ordre, laquelle interviendrait lors de chaque décision pour déterminer la règle valide. Cependant, plusieurs arguments militent en faveur d'une compilation en arbre, en outre, évidemment du coût beaucoup plus réduit du processus décisionnel :

1) Les dimensions de systèmes de règles utilisés en pratique, en particulier dans les applications que nous avons considérées, sont assez faibles pour permettre d'envisager raisonnablement leur ε - optimisation. En Reconnaissance de Formes, le nombre de caractéristiques est de l'ordre de 10 à 20, avec au maximum une centaine de règles. L'ordre de grandeur est encore plus faible pour l'application au "Pattern-matching". Dans le domaine des applications à la programmation, quelques statistiques ont été publiées : sur 113 cas d'utilisation de compilateurs de tables de décision analysés dans [104] , aucun ne comportait plus de dix caractéristiques (binaires) ou de 60 règles. Pour ces ordres de grandeur, même une optimisation exacte par programmation dynamique est faisable.

2) Si l'on s'intéresse à un système de règles dont la dimension reste (même après les réductions proposées de redondance, de coût, et de contraintes de succession) beaucoup trop grande, une compilation partielle demeure avantageuse. Le processus décisionnel démarrera par un arbre de décision opérant une partition aussi fine que possible sur l'ensemble des règles ; les règles de la classe sélectionnée dans une feuille de l'arbre étant alors séparées par une interprétation séquentielle portant sur celles de leurs caractéristiques non encore évaluées.

3) A l'exception de l'algorithme A2 (Programmation Dynamique dont les complexités moyenne et maximale sont similaires), jusqu'à présent, tous les algorithmes ont été anlaysés en pire cas, et tous les problèmes caractérisés par leur complexité maximale. Comme il a été souligné au chapitre 2, les hypothèses de l'analyse en pire cas sont particulièrement pessimistes et défavorables à un schéma

d'approximation tel que AO_ε . Par ailleurs, la non appartenance à NP est basée sur un seul contre-exemple. Il n'est pas exclu que pour une large classe d'applications, les arbres de décision représentant un système ne restent polynômiaux, et les algorithmes d'approximation ne conduisent à des temps de recherche moyens très raisonnables. Un modèle des performances moyennes de l'algorithme AO_ε est, en particulier, absolument nécessaire pour délimiter son domaine d'application pratique.

L'analyse théorique de la complexité moyenne d'un algorithme doit faire face à deux types de problèmes. L'un est dû à la difficulté de définir des distributions de probabilité, significatives pour le type d'applications considérées, sur les instances de données que l'algorithme aura à traiter. L'autre est dû à la difficulté intrinsèque d'analyse du comportement moyen d'un algorithme.[1] Ces deux difficultés sont amplifiées,dans notre cas, par :

1) le niveau élevé de structuration d'un système de règles : caractéristiques explicites et leurs valeurs et caractéristiques indifférentes dans un terme, distribution des termes sur l'ensemble des règles, multiples relations de dépendance et, éventuellement, contraintes de succession et règle Autre, avec de plus les distributions paramétriques de coût et de probabilité. A titre de comparaison, le modèle des graphes aléatoires de Erdös nécessite la donnée d'une fonction unique : p(n), probabilité uniforme avec laquelle un couple de sommets quelconques est une arête d'un graphe de n sommets ; de plus, de nombreux théorèmes caractérisant les propriétés essentielles de ces graphes sont connus (112,113,114,193) , alors qu'aucun résultat de ce type n'est disponible sur les systèmes de règles.

---

(1) Sur cette difficulté, Karp apporte la caution suivante (112) :
"The algorithms we have chosen to analyze are very simple. In every instance heuristics will occur to the reader that would undoubtedly improve the performance of the algorithms ; unfortunately, the introduction of complex heuristics introduces probabilistic dependancies that are extremely difficult to analyze".

2) La complexité conceptuelle importante de l'algorithme AO$_\varepsilon$ qui fait intervenir de nombreuses stratégies : de développement d'un sommet (choix d'un connecteur $\mathcal{J}$ par SELECTION), d'extension d'un SL (sommet suivant à développer par CHOIX), de backtracking (complet, partiel jusqu'à la première bifurcation acceptable, ou intermédiaire), d'exploration de H, et éventuellement, stratégie persévérante. Une condition nécessaire et suffisante de développement d'un sommet de H serait un premier pas dans l'analyse en moyenne de l'algorithme, mais elle ne semble pas facile à trouver.

On a donc réalisé une analyse empirique du comportement moyen du schéma d'approximation, en projetant d'aborder une étude théorique ultérieurement. Le reste de cette section décrit le contexte d'expérimentation, l'implémentation de l'algorithme utilisé et l'analyse des résultats obtenus.

4.3.1. IMPLEMENTATION

Trois implémentations du schéma d'approximation pour processus décisionnels ont été réalisées. La première, orientée vers la traduction des tables de décision, est assez rigide et restreinte, aussi bien au niveau du modèle du système de règles, qu'au niveau des stratégies d'explorations incorporées (1000 lignes de PL1, performances partiellement décrites dans (67)). La deuxième est spécifique à la génération d'arbres d'identification en reconnaissance de forme pour des caractéristiques à distributions paramétriques (75 fonctions APL, décrites au chapitre 5). Moyennant quelques modifications, l'une quelconque de ces deux versions aurait pû être utilisée pour élaborer un modèle du comportement moyen de l'algorithme. Mais, pour des difficultés pratiques (transportabilité du logiciel de CIRCE-Paris au Centre de Calcul de Berkeley), aussi bien que pour pouvoir disposer d'une implémentation orientée dès la conception vers le test de plusieurs variantes de stratégies de recherche, on a préféré en développer une troisième décrite ici.

L'expérimentation a porté sur des systèmes de règles à caractéristiques binaires, sans règle Autre ni contrainte de succession, et sur le critère et les paramètres de coût du modèle de base. Mais, la structure des données et des programmes a été prévue pour permettre d'y incorporer la plupart des extensions mentionnées au chapitre 3 sans trop de difficultés ; certaines de ces extensions, telles que les contraintes de succession ayant été programmées.

Pour disposer de la souplesse et de la facilité de modification du code nécessaires au test de plusieurs stratégies, mais aussi pour des raisons de simplicité et de rapidité d'implémentation, on a adopté APL comme système de programmation. Cela a pratiquement imposé le type de structure de données, les choix de détails ayant été effectués :

- en admettant l'hypothèse d'un nombre relativement faible d'actualisations, et donc en optant pour une structure plus favorable à des descentes en profondeur dans H qu'à des backtracking fréquents ;

- en essayant d'exploiter, au mieux, l'ordre partiel des rangs sur H ; et

- en fixant le compromis "information apportée par un pointeur - coût de gestion de ce pointeur" en faveur d'un parcours plus rapide sur le SL Λ en cours de développement, que sur le reste de l'espace H.

Cela conduit à définir le $i^{ème}$ sommet u créé dans H par :

- le r-cube (u(1), u(2), ..., u(n)) ;
- les deux listes d'indices $\Phi(u)$ et $\{1, \ldots n\} - \Phi(u)$;
- les coûts cumulés sur les $|\Phi(u)|$ connecteurs : $g_j(u) = \rho_j\, q_j(u) + \sum_{v \in \Gamma_j(u)} g(v)$;
- l'indice du connecteur $\hat{j}(u)$;
- un pointeur vers w tel que $u \in \Gamma_{\hat{j}}(w)$: père unique de u dans Λ ;
- des pointeurs vers $\Gamma_{\hat{j}}(u)$: ensemble des fils de u dans Λ ;
- la fraction de valeur de g(u) non encore reportée par une actualisation sur tous les pères de u.

Chacun de ces ensembles de données peut être adressé dans une table particulière par la seule connaissance de l'indice i.

L'ordre partiel des rangs sur H est explicitement retenu en associant à chaque rang k la liste de tous les sommets de ce rang connus dans H (i.e., pendants ou développés).

Au SL Λ courant correspondent deux piles FIFO qui rendent triviale la procédure CHOIX :

NP : liste des sommets de Λ de rang k dont un des fils dans Λ est pendant ;

NQ : liste des sommets de Λ de rang k dont tous les fils dans Λ sont développés.

La définition de Λ après une actualisation (SOUS - LATTICIEL) consiste à examiner tous les sommets de NQ, en répartissant leurs fils entre les listes NP et NQ du rang suivant ; on initialise à NQ = $\{u_o\}$, et on arrête au premier rang k pour lequel NP est non vide. Les développements de Λ procèdent alors en prenant le premier sommet de la liste NP, en développant ses fils pendants et en les répartissant entre les listes du rang suivant ; Λ est complet lorsque NP est vide.

Cette structure facilite également une actualisation partielle qui ne remonte pas le backtracking jusqu'à la racine u_o, mais procède par des modifications ascendantes de H, complètes par rang, jusqu'au premier rang pour lequel le Λ obtenu est de nouveau acceptable.

On a exploité la plupart des propriétés et simplifications décrites en (3.2.3.), en particulier les suivantes :

- élimination des caractéristiques complètement redondantes dans un sous-système S/u par l'intermédiaire de pénalisations sur les estimations de coût des connecteurs correspondants ;

- développement en profondeur systématique pour les caractéristiques nécessaires ;

- test si un sommet est terminal, effectué uniquement lors du développement de ce sommet ;

- procédure de développement simplifiée pour les sommets u tel que $|X/u| = 1$.

Le logiciel complet est relativement concis. Il comporte 22 fonctions APL structurées en :

- fonctions d'entrée-sortie intéractives ;
- générateur de données pseudo-aléatoires ;
- prétraitement d'un système de règles ;
- algorithme AO_ϵ ;
- gestion de l'expérimentation pour la construction du modèle de comportement moyen de l'algorithme.

On suppose que le système de règles S, fourni par l'utilisateur ou généré aléatoirement, est sous forme normale, consistant et complet ; et que la distribution de probabilité donne explicitement $\mu(u)$ pour tous les termes u de S, les états couverts par u étant supposés équiprobables. Moyennant ces hypothèses, le prétraitement de S se réduit à l'algorithme RECOUVREMENT et fournit pour chaque caractéristique X_j les ensembles C_j, C'_j et C''_j sous forme d'une table unique comportant moins de $m = \sum_{i=1}^{a} |R_i|$ termes.

Le calcul de :

$$\rho_j \, q_j(u) = \rho_j \sum_{v \in C_j \cup C'_j \cup C''_j} \mu(v \cap u)$$

et la détermination de la redondance complète se font par un parcours de cette table au pire en $O(mx(n - |\Phi(u)|))$.

Le générateur de données pseudo-aléatoires utilise l'opérateur d'APL "?" de tirage uniformement distribué sur un segment d'entiers (1), et correspond à l'algorithme suivant :

---

(1) Opérateur du type congruentiel : y ← ? k affecte à y un entier entre 1 et k par : $x_n \leftarrow (x_{n-1} \times 7^5)$ modulo $(2^{31} - 1)$, suite initialisée par le système à 7^5, et $y \leftarrow 1 + \lfloor k \times x_n / (2^{31} - 1) \rfloor$.
Comme tous les générateurs de ce type, les distributions sur un ensemble de séquences sans répétitions sont biaisées. On y a remédié en appliquant à chaque séquence générée une permutation aléatoire (Cf. Knuth (122) et, récemment Herzog (90)).

Algorithme GENERATION

Données d'entrée : 3 entiers n, δ, et Δ

1. Prendre m uniformément distribué dans $\{n, n+1, \ldots, 3n\}$

2. Prendre a uniformément distribué dans $\{2, 3, \ldots, m\}$

3. Si $\delta \neq 0$, alors $d \leftarrow \lceil n \times 2^{\delta + 1-n} \rceil$

   Sinon $d \leftarrow 0$

4. $T \leftarrow \{ u_o = (\varphi, \varphi, \ldots, \varphi) \}$

5. Itérer tant que T comporte moins de (m+d) termes
   Prendre aléatoirement un terme u non élémentaire dans T et le supprimer de T
   Prendre aléatoirement un indice i dans $\Phi(u)$
   Mettre dans T les 2 termes $(u/i \leftarrow 0)$ et $(u/i \leftarrow 1)$
   Fin

6. Si $d \neq 0$, alors prendre aléatoiremnt d termes $\{u_1, \ldots u_d\}$ dans T tel que $2^{\delta-1} \leq \sum_{i=1}^{d} 2^{|\Phi(u_i)|} \leq 2^{\delta+1}$, et affecter ces termes à l'ensemble D en les supprimant de T

7. Répartir aléatoirement les m termes restants de T entre a règles, avec au minimum 1 terme par règle

8. Générer n entiers $\rho_1, \ldots, \rho_n$ tel que ρ_i est uniformément distribué dans $\{1, \ldots, \Delta\}$

9. Générer m entiers $\mu_1, \ldots \mu_m$ tel que μ_i est uniformément distribué dans $\{1, \ldots, \Delta\}$

Données de sortie : un système de règles S de n caractéristiques binaires, de d termes de dépendance, de a règles comportant un total de m termes, et admettant la distribution de coût $(\rho_1, \ldots, \rho_n)$ et la distribution de probabilité $(\mu_1/\Sigma\mu_i, \ldots, \mu_m/\Sigma\mu_i)$.

Le nombre m des termes des règles de S a été pris entre n et $3n$ en se référant aux observations empiriques sur l'utilisation des tables de décision qui notent, en général, un nombre de termes de l'ordre de 2n (plutôt que la limite maximale de 2^n pour n caractéristiques binaires).

Les bornes pour le nombre a des règles de S ont été prises à leurs valeurs extrêmes : 2 et m. Le paramètre d'entrée δ représente le "degré de dépendance" des caractéristiques du système qu'on veut générer : à peu près 2^{δ} sommets élémentaires de n-cube $\{0, 1\}^n$ seront couverts par D et correspondront à des états impossibles. Le nombre d de termes de D est pris en admettant une distribution uniforme des φ dans les termes. Le troisième paramètre d'entrée Δ représente le degré de dispersion des distributions de coût et de probabilité. Dans toutes les expérimentations effectuées, on a borné à 50 le rapport entre les valeurs max et min de ces distributions.

On vérifie facilement que le système S généré est consistant et complet et que deux quelconques de ses (m+d) termes sont sans recouvrement. Cependant, même en admettant que l'opérateur "?" d'APL est statisquement parfait, le générateur de données est structurellement biaisé : les (m+d) termes de S ne constituent pas une partition quelconque du n-cube, mais une partition définissable par un arbre.

La raison de cette restriction aux partitions de Π_T réside principalement dans la simplicité du générateur (m+d itérations seulement). Par ailleurs, on a estimé (et l'expérimentation le confirme) que ce biais a un impact négligeable sur les performances de l'algorithme qu'on cherche à modéliser : vu du côté de AO_ε, seule compte la partition en (a+1), blocs définie par les a règles et l'ensemble D. Les (m+d) termes étant répartis aléatoirement entre ces (a+1) blocs, cette partition est de nouveau quelconque (ni homogène, ni dans Π_T), et des regroupements peuvent s'effectuer entre les termes des ensembles ($R_i \cup D$) selon l'une quelconque de leurs coordonnées sans privilégier aucune caractéristique. Une seule exception cependant : pour S sans dépendance et ayant exactement 1 terme par règle (d = 0 et a = m), l'arbre correspondant à la partition des m termes de S est toujours optimal, et il est trouvé par AO_ε par une simple recherche en profondeur en O(m). Ce cas là fut éliminé par la suite.

4.3.2. EXPERIMENTATION

Elle a consisté à :

- choisir un couple de paramètres (n, δ) ;
- générer aléatoirement les systèmes de règles de n caractéristiques de degré de dépendance δ, et pour chacun, générer l distributions de coût et de probabilité (avec $\Delta = 50$) ;
- appeler 5 fois l'algorithme AO_ε sur chacun des (k x l) jeux de données avec ε fixé successivement à[1] : $\varepsilon = \infty$; $\varepsilon = 0.1$; $\varepsilon = 0.05$; $\varepsilon = 0.01$; et $\varepsilon = 0$.

Chaque appel de AO_ε entraîne l'enregistrement d'un point de mesure consistant en les 11 paramètres (n, m, a, d, ε, ψ_r, ε', G, D, B, T, I) suivants :

- n, m, a, d, ε : paramètres du système de règles traité dans l'essai ;
- $\psi_r = \psi(t)/(\sum_{i=1}^{n} p_i)$: coût relatif de la solution trouvée ;
- ε' : valeur retournée par l'algorithme du majorant de l'écart relatif à l'optimum ;
- G : nombre total de sommets de H (pendants ou développés) générés ;
- D : nombre total de sommets de H développés ;
- B : nombre total d'actualisations effectuées ;
- T : temps Cpu mis pour atteindre la solution ;
- I : nombre total d'opérateurs APL interprétés durant l'essai.

Un total de 3039 essais furent effectués pour des systèmes de règles ayant entre 4 et 15 caractéristiques, et de 2 à 41 règles. Les résultats ont été analysés grâce aux programmes du logiciel statistique STATPACK (IBM).

L'expérimentation a démarré sur un DEC.20 au Centre de Calcul de l'Université de Californie à Berkeley, puis s'est poursuivie successivement sur :

- un IBM 370.168 (CIRCE - Paris) ;

---

(1) La notation $\varepsilon = \infty$, utilisée par commodité, correspond en pratique à la valeur $\varepsilon = (\sum_{i=1}^{n} p_i)/h(u_0)$ qui assure exactement le même comportement de l'algorithme que la valeur théorique $\varepsilon = \infty$.

- un AMDAHL V7 (CIRCE - Paris) ;
- un IBM 3033 (CNUSC - Montpellier).

Des précautions ont été prises pour ramener les mesures de complexité à des variables algorithmiques, de façon à garder des résultats comparables malgré cette séquence de déménagements (en partie imprévue). Une analyse de 851 points de mesure effectués à Berkeley a conduit aux observations suivantes :

- l'indicateur I (disponible uniquement sur l'APL-SF de DEC ; mesure globalement le nombre d'opérateurs interprétés indépendamment de la taille des opérandes) est corrélé au temps CPU avec un coefficient de corrélation $r \geqslant 0.99$; T est cependant moins fiable, car pour le même calcul, il peut varier de façon significative avec la charge de la machine. I a donc été retenu comme premier indicateur de la complexité ;

- les variables G et D sont totalement corrélées ($r = 1$) ; D a été retenu ;

- sur 330 essais avec $\varepsilon = \infty$ (i.e., sans backtracking : B = 0) l'indicateur I a été trouvé linéaire en le produit n x D avec une corrélation : $r = 0.99$;

- divers polynômes en n; D et B ont été testés sur les 851 points de mesures, la corrélation maximale donne I linéaire en n x D et en n x B. Numériquement, le maximum correspond à $Y = n\,(D + 2.3 \times B)$ avec une corrélation $r = 0.97$.

On a donc retenu Y(n, D, B) comme l'indicateur algorithmique de complexité indépendant de la machine utilisée. Testée par la suite, la validité de Y s'est révélée satisfaisante :

- sur 561 points de mesure effectués sur l'AMDAHL, T (temps Cpu) et Y sont corrélés avec $r = 0.97$;

- sur 937 mesures effectués sur l'IBM 3033, la corrélation a été de $r = 0.93$.

Pour des raisons de coût d'expérimentation, le nombre de points de mesures enregistrés est décroissant avec n. De plus, à partir de $n \geqslant 9$, la séquence d'essais a été restreinte à $\varepsilon \in \{\infty, 0.1, 0.05\}$, et pour n = 14 et n = 15, on s'est

contenté de $\varepsilon = \infty$. Par ailleurs, dans de nombreux cas (surtout pour n faible), la valeur de ε' retournée par l'algorithme était inférieure à la valeur suivante de ε, ce qui contraignait à omettre les (ou la) mesures suivantes.

Toutes les mesures effectuées n'ont pas été retenues pour l'élaboration du modèle de complexité. Les points éliminés correspondent, en très grande majorité, au problème du ε' - inférieur aux ε suivants, et aux mesures manquantes. Certains de ces cas étaient dûs au biais du générateur de données : quelles que soient les distributions de paramètres sur un système de règles, l'algorithme trouve toujours la solution optimale ($\varepsilon' = 0$) dès le premier appel avec $\varepsilon = \infty$. N'ont pas été retenus également, les systèmes comportant initialement plus de deux caractéristiques complètement redondantes.

Les éliminations précédentes correspondent à des valeurs de l'indicateur de complexité y très faibles par rapport à la moyenne. On a hésité à effectuer l'élimination symétrique (du type : suppression des mesures à plus de 3 écarts types de la moyenne). Il s'est avéré que les points de complexité la plus élevée correspondent au problème de la dimension exponentielle de l'arbre de décision généré. On a estimé que cela représentait une des composantes de la complexité à modéliser, et seuls 4 points aberrants n'ont pas été retenus (par exemple : système de 14 caractéristiques non redondantes et de 2 règles, pour une des distributions de paramètres l'arbre solution à $\varepsilon = \infty$ comportait 2155 noeuds, alors que la moyenne sur ce même système pour d'autres distributions était de l'ordre d'une centaine de noeuds).

Finalement, la table ci-dessus fournit le nombre de points de mesure retenus et sur lesquels s'appuie le modèle qui va suivre :

n	4	5	6	7	8	9	10	11	12	14	15
nombre d'essais	197	307	254	347	168	147	35	9	24	24	9

4.3.3. MODELE ET TEST DE STRATEGIES D'EXPLORATION

Une analyse quantitative de données expérimentales devrait viser, dans notre cas, les objectifs suivants :

- modélisation de l'indicateur de complexité $Y = n\ (D + 2.3\ B)$ par une fonction $\hat{Y}$ des paramètres des données d'entrée : ε , n, m, a, d ;

- modélisation du majorant ε' de l'écart à l'optimum fourni par l'algorithme, et de l'écart réel à l'optimum, ε_r , par des fonctions de ε , n, m, a, d ;

- modélisation du coût relatif $\Psi_r = \Psi(t)/(\Sigma \rho_i)$ par une fonction de ε , n,m,a,d.

A cause de l'insuffisance des essais effectués par rapport aux dimensions du problème, ces objectifs n'ont été que très partiellement atteints. En fait, le quintuplet (ε, n, m, a, d) n'est constant que sur un nombre extrêmement faible de points de mesure, lesquels peuvent être considérablement dispersés (dans un rapport 20 sur le même système de règles). Une agrégation par rapport aux paramètres les moins significatifs était donc nécessaire. La modélisation a été finalement restreinte à Y en fonction de n et de ε , en se contentant d'une analyse qualitative pour les autres paramètres.

Les courbes (figures 4.5. et 4.6) représentent les valeurs moyennes de Y tracées respectivement en fonction de n et de ε (tout au long de cette section, ε sera exprimé en pourcentage d'écart relatif) ; l'écart type de Y étant indiqué sur quelques points par une barre verticale.

Ces courbes suggèrent des modèles du type :

$\hat{Y}(n,\varepsilon) = \alpha(\varepsilon) + \beta(\varepsilon) \times \gamma(\varepsilon)^n$; ou bien

$\hat{Y}(n,\varepsilon) = a(n) + b(n) \times c(n)^{-\varepsilon}$.

Des modèles polynômiaux furent également essayés, mais les meilleurs résultats ont été obtenus par une formulation exponentielle.

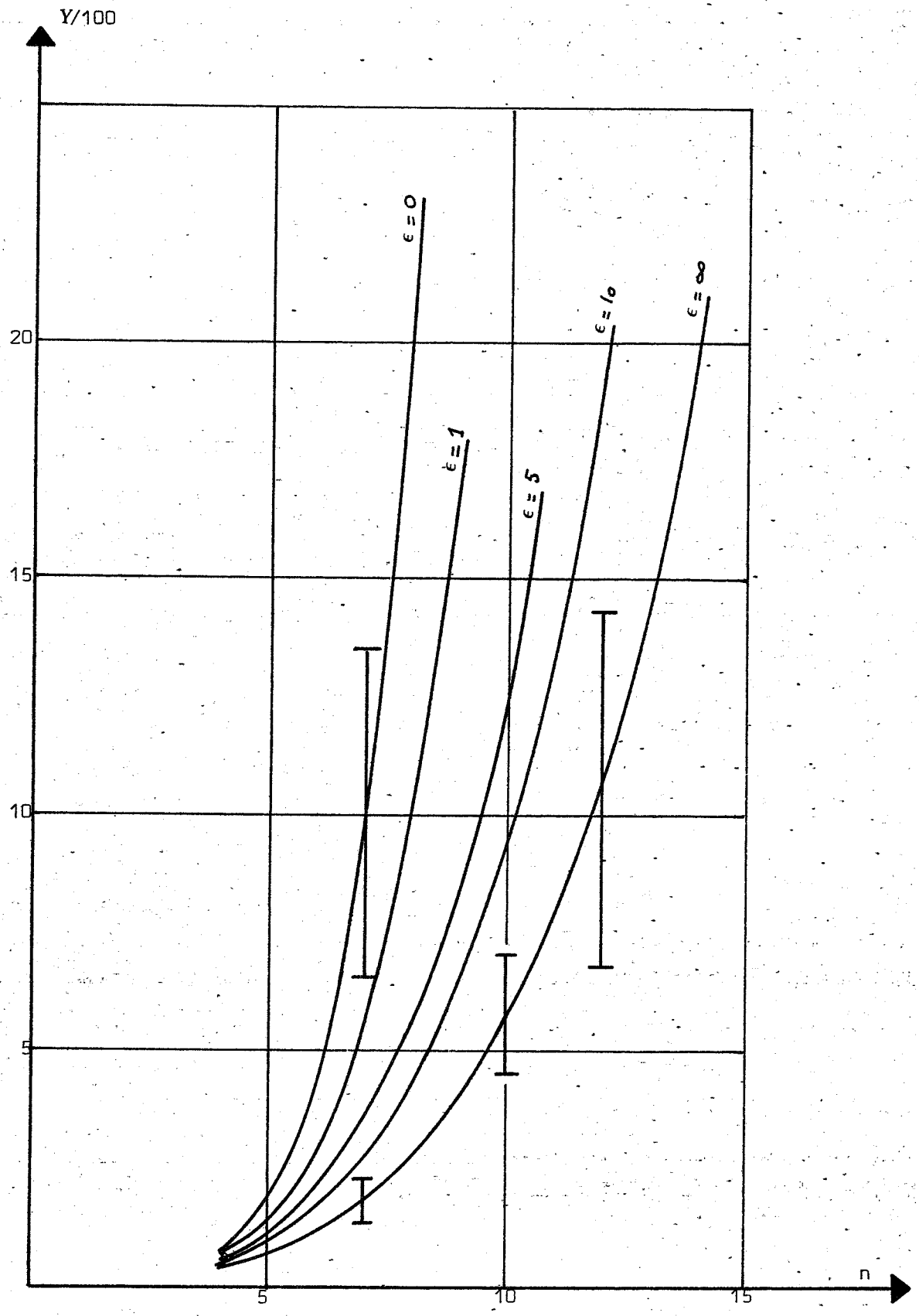


FIGURE 4.5.

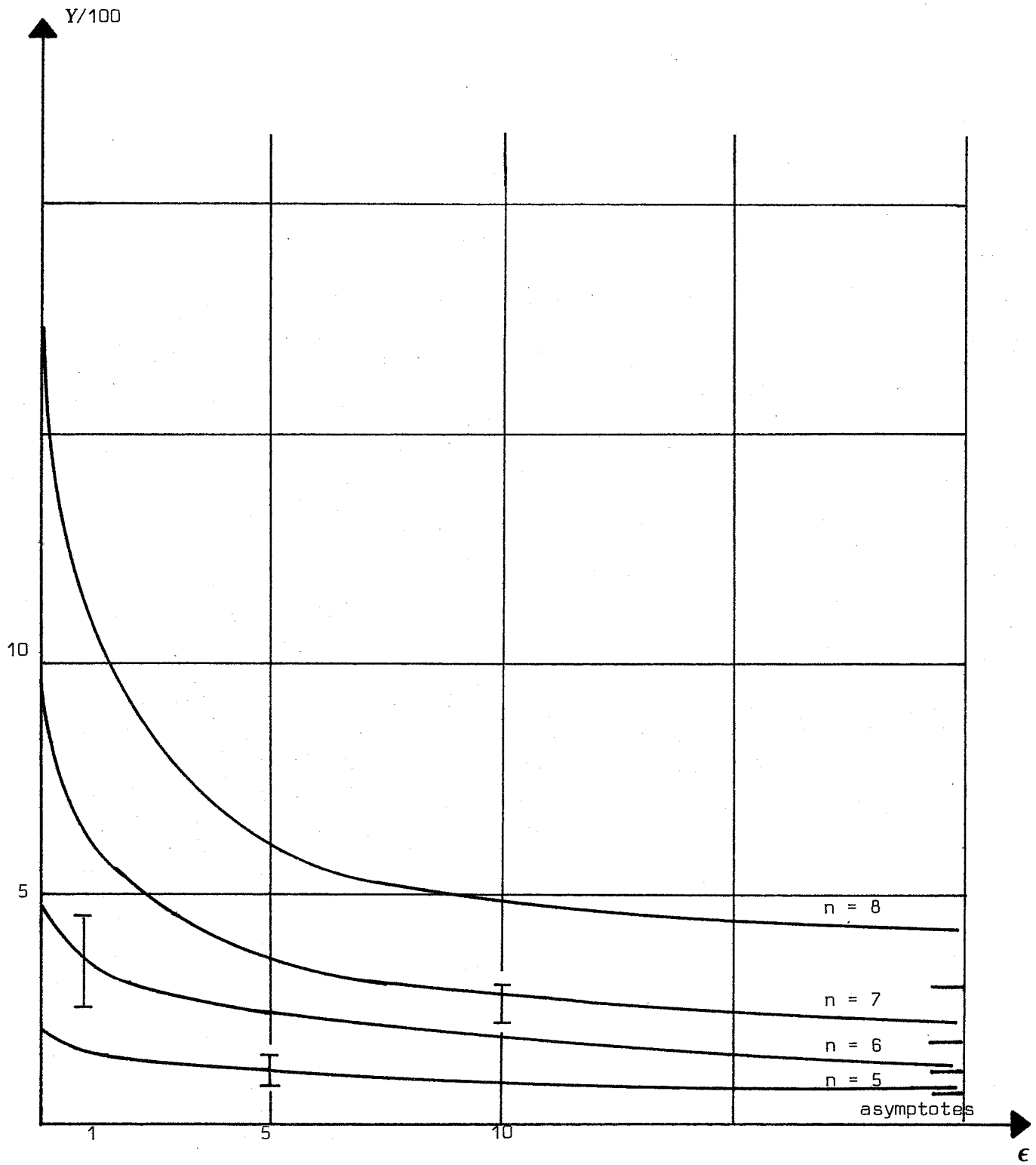


FIGURE 4.6.

La démarche menant au modèle finalement retenu a été la suivante :

- les régressions entre Y et γ^n, pour les valeurs de γ correspondant à une corrélation maximale, donnent :

$\hat{Y}\ (\varepsilon = 0) = -57.6 + 8.36 \times 1.94^n + N(0\ ;\ 14)$, avec $r = .999$

$\hat{Y}\ (\varepsilon = 1) = -123 + 25.91 \times 1.62^n + N(0\ ;\ 8.9)$, avec $r = .999$

$\hat{Y}\ (\varepsilon = 5) = -11 + 21.04 \times 1.50^n + N(0\ ;\ 44)$, avec $r = .995$

$\hat{Y}\ (\varepsilon = 10) = -30 + 22.62 \times 1.46^n + N(0\ ;\ 45)$, avec $r = .998$

$\hat{Y}\ (\varepsilon = \infty) = -137 + 45.27 \times 1.32^n + N(0\ ;\ 88)$, avec $r = .995$

$N(0\ ;\ \sigma)$ étant une erreur gaussienne à moyenne nulle et à écart type σ.

- la courbe traçant γ en fonction de ε (figure 4.7.) suggère une décroissance exponentille, que la régression donne numériquement égale à :

$\gamma\ (\varepsilon) = 1.42 + 0.57 \times 2.86^{-\varepsilon} + N(0\ ;\ 0.07)$.

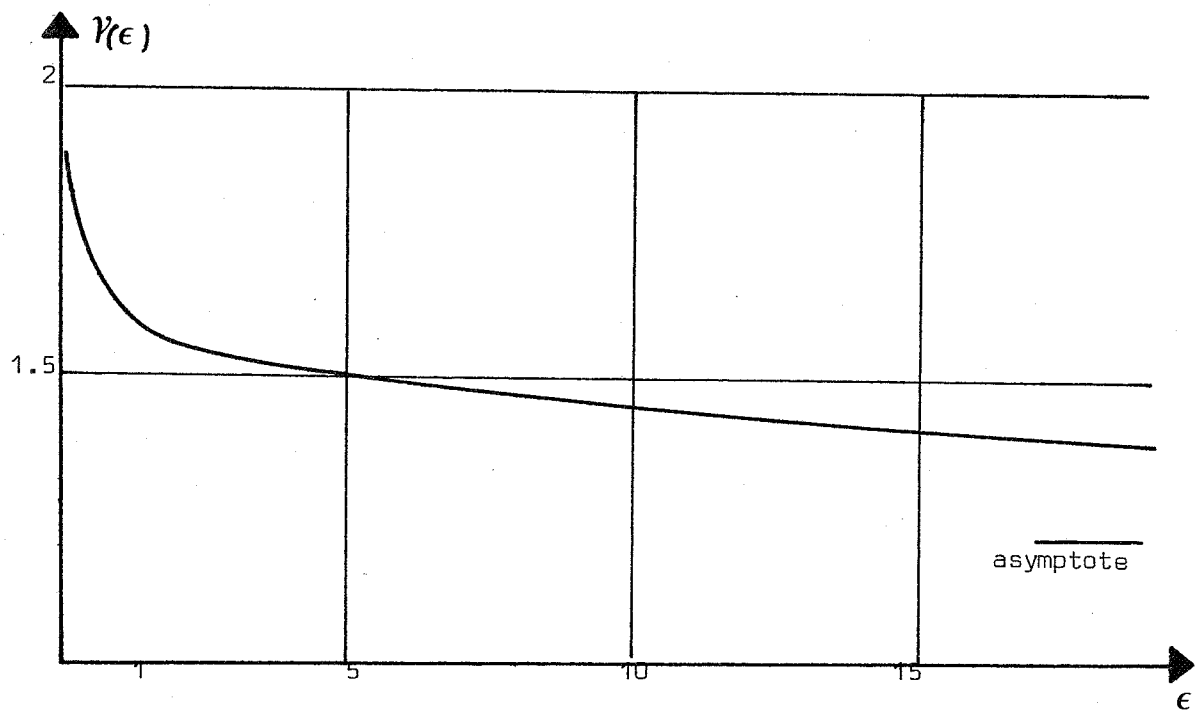


FIGURE 4.7.

- sur ces résultats, les distributions de α et de β , pour une formulation du type $\hat{Y}(n,\varepsilon) = \alpha(\varepsilon) + \beta(\varepsilon) \cdot \gamma(\varepsilon)^n$, ne conduisent pas à un modèle simple satisfaisant. On peut alors remarquer que :

  . les valeurs de $\gamma(\varepsilon)$ correspondent à des maxima du coefficient de corrélation extrêmement plat : un $\Delta\gamma = 0.1$ entraîne un $\Delta r = 10^{-3}$;

  . en dehors de $b(\varepsilon = 0)$ et $b(\varepsilon = \infty)$, $b(\varepsilon)$ est pratiquement constant entre 21 et 25.

Aussi, a-t-on légèrement modifié $\gamma(\varepsilon)$ dans le voisinage des valeurs précédentes pour se ramener à une formulation $\hat{Y} = \alpha(\varepsilon) + \beta \cdot \gamma(\varepsilon)^n$; β étant constant. Les régressions conduisent à :

$$\alpha(\varepsilon) = -27.3 - 152.6 \times 1.845^{-\varepsilon} + N(0 ; 2.33)$$

$$\beta = 22$$

$$\gamma(\varepsilon) = 1.41 + 0.36 \times 1.37^{-\varepsilon} + N(0 ; 0.037)$$

- une analyse de cette formulation montre que la discontinuité importante entre $Y(\varepsilon = 1)$ et $Y(\varepsilon = \infty)$ est improprement modélisée. Cette discontinuité se remarque facilement sur les figures 4.6. et 4.7. : les valeurs $Y(\varepsilon = \infty)$ et $\gamma(\infty)$ sont très inférieures aux asymptotes des courbes exponentielles. En fait $\gamma(\infty) = 1.41$ conduit à une croissance de $\hat{Y}(\varepsilon = \infty)$ divergente par rapport aux valeurs mesurées de Y. Une formulation tenant compte séparément de $\varepsilon = \infty$ serait plus précise. En reprenant les régressions sur cette base, on obtient :

$$\hat{Y}(n,\varepsilon) = \alpha(\varepsilon) + 22 \times \gamma(\varepsilon)^n \quad \text{pour } \varepsilon < \infty \text{ , avec :}$$

$$\alpha(\varepsilon) = -45 - 105 \times 1.72^{-\varepsilon}$$

$$\gamma(\varepsilon) = 1.465 + 0.315 \times 1.65^{-\varepsilon} \text{ ; et}$$

$$\hat{Y}(n,\infty) = -105 + 43.3 \times 1.32^n$$

Les trois tables ci-dessous donnent respectivement les moyennes des valeurs mesurées de $Y(n,\varepsilon)$; les valeurs prédites $\hat{Y}(n,\varepsilon)$ par ce modèle ; et les écarts relatifs en pourcentage : $100 \times |\hat{Y} - Y| / Y$. On constate que l'écart maximal est de 28.2 %, alors que l'écart moyen est de 8.3 %.

Y

n \ ε	0	1	5	10	∞
4	70.853	59.366	56.68	56.459	26.457
5	243.12	167.86	110	104.07	68.523
6	549.75	347.52	189.5	173.91	124.05
7	1095.6	645.02	308	276.39	197.35
8	2067.1	1137.7	484.66	426.72	294.1
9			748.01	647.29	421.81
10			1140.6	970.87	590.39
11					812.91
12				2142.1	1106.6
14					2006.2
15					2681.7

$\hat{Y}$

n \ ε	0	1	5	10	∞
4	71.054	58	52.533	44.022	29.394
5	207.81	157.35	115.25	99.621	65.649
6	497.3	356.35	255.35	199.56	113.1
7	998.92	629.95	354.36	297.09	173.83
8	2088.3	1107.6	596.67	513.18	292.67
9			806.62	659.7	336
10			1166.5	896.73	580
11					827.44
12				2104.7	1046
14					2279.7
15					2665

$100 \times \frac{|\hat{Y}-Y|}{Y}$

n \ ε	0	1	5	10	∞
4	0.28284	2.3554	7.8934	28.251	9.9917
5	16.992	6.6791	4.5563	4.463	4.3776
6	10.547	2.4765	25.789	12.851	9.6823
7	9.6734	2.3919	13.084	6.9693	13.526
8	1.0158	2.7131	18.773	16.847	0.489
9			7.2663	1.8817	25.538
10			2.2192	8.2686	1.791
11					1.7563
12				1.777	5.7977
14					11.999
15					0.62824

Une démarche similaire peut être effectuée pour la formulation symétrique en n et ε . On obtient :

$\hat{Y}(n, \varepsilon) = a(n) + b(n) \times c(n)^{-\varepsilon}$, pour $\varepsilon < \infty$, avec :

$a(n) = -80.7 + 32.1 \times 1.42^{n} + N(0 ; 16.3)$;

$b(n) = -49.11 + 3 \times 2.21^{n} + N(0 ; 5.2)$;

$c(n) = 1.46 + 0.009 \times 1.55^{n} + N(0 ; 0.012)$, et

$\hat{Y}(n, \infty) = -105 + 43.3 \times 1.32^{n}$

Comme le montrent les deux tables suivantes (donnant respectivement les valeurs prédites et les écarts relatifs), ce modèle est légèrement plus précis que le précédent ; en particulier pour $\varepsilon = 5$ et $\varepsilon = 1$.

L'écart maximal est de 25.7 %, alors qu'en moyenne, il n'est que de 7.2 %. Notons également que pour $\varepsilon = 0$, l'ordre de croissance de $\hat{Y}$ est plus élevé dans ce modèle que dans le précédent (2.21^{n}, au lieu de 1.78^{n}).

$\hat{Y}$

n \ ε	0	1	5	10	∞
4	73.374	65.766	53.757	51.279	26.457
5	214.95	176.65	118.44	105.35	68.523
6	484.4	373.38	213.91	186.97	124.35
7	1018.2	731.77	353.02	299.54	197.35
8	2110.3	1392.7	549.61	458.02	294.1
9			814.96	681.07	421.81
10			1158.5	996.64	590.39
11					812.91
12				2084.6	1106.6
14					2006.2
15					2681.7

$100 \times \frac{|\hat{Y}-Y|}{Y}$

n \ ε	0	1	5	10	∞
4	3.266	13.389	2.33	16.485	9.9917
5	3.4385	12.265	2.7634	7.7561	4.3776
6	2.5934	4.7807	16.229	6.3095	9.6823
7	1.9295	16.162	0.37955	0.82396	13.526
8	1.0535	25.736	7.888	10.749	0.489
9			1.0328	3.2396	25.538
10			0.68672	11.141	1.791
11					1.7563
12				0.95296	5.7977
14					11.999
15					0.62824

Finalement, on peut retenir comme conclusion sur l'analyse de Y que la complexité moyenne de l'algorithme AO_ε est en $O(\gamma(\varepsilon)^n)$ avec :

- $\gamma(\varepsilon = 0) \simeq 2$, alors que la complexité moyenne de la Programmation Dynamique est en $O(3^n)$, et que l'analyse au pire des cas donne AO_ε en $O(5^n)$;

- $\gamma(\varepsilon)$ décroît exponentiellement avec ε (ce que ne montre pas l'analyse au pire des cas), ainsi l'ordre de complexité est très sensiblement réduit même pour des valeurs faibles de ε ;

- la décroissance de $\gamma(\varepsilon)$ s'accompagne d'une discontinuité à $\varepsilon = \infty$ qui conduit, pour la recherche sans backtracking, à une complexité exponentielle en $O(1.32^n)$ assez modérée ;

- la complexité moyenne augmente avec le nombre total de termes du système (m et d), mais un grand nombre de règles (a) ou de sommets élémentaires impossibles (2^δ) entraîne plutôt sa diminution.

Abordons maintenant l'analyse des trois autres grandeurs ε', ε_r et ψ_r mesurées durant l'expérimentation.

La table suivante donne le coût relatif moyen ψ_r en fonction de n et de ε . On y remarque que :

- les variations de ψ_r en fonction de ε ne sont pas très significatives et d'un ordre de grandeur bien inférieur à ε ;

- $\psi_r(n)$ est une courbe (figure 4.8 pour $\varepsilon = \infty$) décroissante à peu près linéairement. La droite de régression est :

  $\psi_r(n,\infty) = 0.74 - 0.023 \times n + N(0 ; 0.03)$ avec $r = 0.92$.
  Sachant que $\psi_r(t) = (\sum_{i=1}^{n} p_i) / \psi(t)$, un arbre de coût relatif $\psi_r(t) = 1$ possède $(2^n - 1)$ noeuds (tous ses chemins de la racine à une feuille rencontrent les n caractéristiques). Or, on constate que la taille moyenne des arbres générés n'augmente pas en 2^n bien qu'elle soit exponentielle en n (c'est ce que reflète $Y(n,\infty)$). Cela est dû, en partie, au fait d'avoir pris m et a bornés linéairement en n, et explique pourquoi ψ_r décroît avec n.

Ψ_r

n \ ε	0	1	5	10	∞
4	0.61974	0.6293	0.66213	0.59466	0.60493
5	0.68017	0.67591	0.67064	0.65533	0.67434
6	0.62567	0.62563	0.61673	0.5942	0.63968
7	0.56116	0.55026	0.54454	0.54543	0.56593
8	0.57209	0.57286	0.58687	0.60602	0.58941
9			0.492	0.48447	0.50285
10			0.42896	0.42819	0.44548
11					0.48966
12				0.4413	0.45497
14					0.44096
15					0.40759

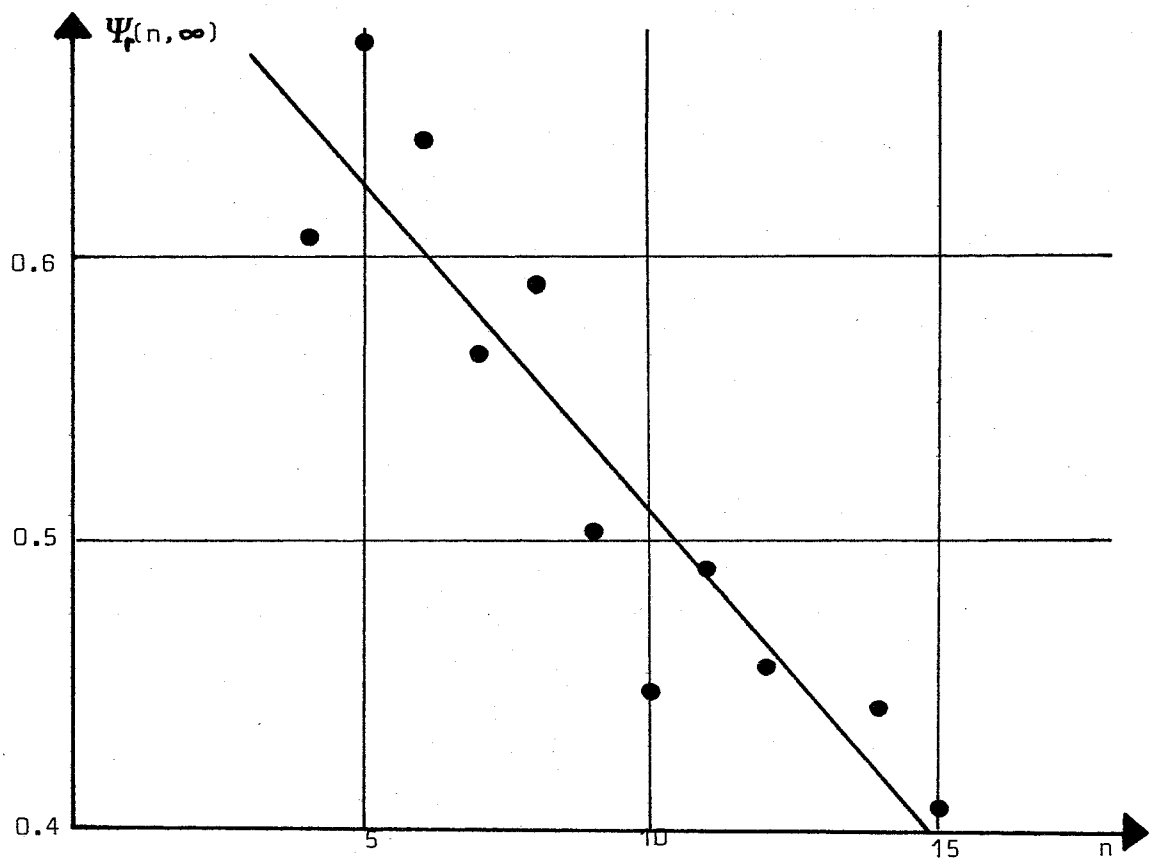


FIGURE 4.8.

L'analyse de la sensibilité de Y aux paramètres m, a et d donne les résultats suivants :

- le coefficient de corrélation partiel de Y et de m pour n,a,d et ε constants, est de l'ordre de + 0.4. On peut attribuer la croissance de la complexité en fonction de m esssentiellement au temps de calcul plus long de la fonction d'évaluation f, car la combinatoire de la recherche ne dépend pas du nombre plus ou moins grand des termes des règles ;

- la corrélation partielle de Y et de a pour n,m,d et ε constants, est négative et de l'ordre de - 0.25. Cela est vraisemblablement dû au fait qu'un nombre de règles plus élevé réduit les regroupements entre termes ; ce qui entraîne davantage de caractéristiques nécessaires ($q_j = 0$) et diminue la combinatoire de la recherche. Une interprétation prudente relèverait également, dans cette corrélation négative, l'influence du biais du générateur de données (malgré l'élimination des cas extrêmes a = m et d = 0) ;

- la corrélation partielle de Y et de d est trop faible pour être significative. De plus, elle passe d'une corrélation positive pour ε = 0,1 ou 10, à une corrélation négative à ε = 5 ou ∞. En fait, l'analyse de l'algorithme montre que les relations de dépendances ont deux influences contradictoires :

  . d'une part, elles augmentent considérablement les regroupements possibles entre termes, et donc le temps de calcul de la fonction d'évaluation ;

  . d'autre part, elles réduisent à chaque développement le nombre de choix possibles ($|X/u|$ est plus faible), et diminuent la dimension des arbres de décision représentant un système de règles (du fait de l'élagage des branches impossibles) ; ce qui entraîne une réduction de la combinatoire de la recherche.

La première influence devrait être prépondérante pour d grand et petit ; un rapport inverse étant favorable à la deuxième. Les hypothèses dans le générateur de données ($d = 1 + \lceil n \times 2^{\delta} + 1 - n \rceil$) ne permettent pas de séparer ces deux influences.

Le majorant du degré d'optimalité retourné par l'algorithme qui a été mesuré tout au long de l'expérimentation n'est pas $\varepsilon' = \max\left\{\frac{Y-y}{y}, 0\right\}$ en fin d'exploration, mais plutôt $\varepsilon'' = \max\left\{\max \frac{Y-y}{y}, 0\right\}$, i.e., écart relatif correspondant à la valeur de F, avant backtracking, la plus proche du seuil $(1+\varepsilon)y$, qui ait été enregistré tout au long de la recherche. On a bien sûr $\varepsilon' \leqslant \varepsilon'' \leqslant \varepsilon$; et tous les appels de AO_ε avec des paramètres compris entre ε'' et ε conduisent exactement aux développements et actualisations des mêmes sommets dans le même ordre. Cette propriété justifie l'utilité de ε'' pour l'expérimentation (il permet d'omettre des essais identiques), mais son enregistrement dans chaque point de mesure à la place de ε' résulte en fait d'une erreur qui n'a été détectée qu'à l'analyse des résultats (les chances de passer "très près" du seuil $(1+\varepsilon)F'$ augmentent avec le nombre de backtrackings, et l'écart moyen entre ε'' et ε devient insignifiant pour n grand et ε petit). Cependant, pour un appel de AO_ε sans backtracking ($\varepsilon = \infty$), l'erreur précédente est sans conséquence, car ε'' et ε' sont identiques.

La table suivante donne les moyennes des valeurs enregistrées $\varepsilon'(n, \infty)$. On note une croissance quasi-linéaire en fonction de n, la droite de régression étant :

$\varepsilon'(n, \infty) = 0.043 + 0.015 \times n + N(0 ; 0.058)$; avec $r = 0.7$

n	$\epsilon'(n, \epsilon = \infty)$
4	0.09253
5	0.10552
6	0.13084
7	0.15947
8	0.11798
9	0.18787
10	0.19375
11	0.32304
12	0.18842
14	0.3169
15	0.17138

On note également que l'ordre de grandeur de ε' est tout à fait satisfaisant : un écart à l'optimum meilleur que 15 à 20 % (pour $\varepsilon = \infty$) constitue une garantie très appréciable sur le plan pratique.

Cette remarque peut être renforcée par le fait que les mesures enregistrées de l'écart réel à l'optimum, $\mathcal{E}_r$, sont très dispersées entre 0 et $\mathcal{E}'$. En moyenne, $\mathcal{E}_r$ n'est cependant que de l'ordre de $\mathcal{E}'/10$ (table suivante).

$100 \times \mathcal{E}_r(n,\infty)$

$\mathcal{E}$ \ n	4	5	6	7	8	9
∞	1.6	1.8	2.3	3.15	3.3	3.5
1	.5	.7	.83	.75	1.5	

Le dernier aspect de ce travail de modélisation empirique porte sur le test de plusieurs stratégies d'exploration. Sur l'ensemble des variantes mentionnées dans les chapitres 2 et 3, seules deux stratégies, en plus de celles décrites jusqu'à présent, ont été expérimentées (expérimentation effectuée à Berkeley ; on continue à projeter de la poursuivre sur un problème à combinatoire beaucoup plus réduite que celle de l'optimisation des PDF).

La première stratégie testée (S1) est celle du backtracking "court" : l'actualisation ascendante des sommets de H s'arrête dès le premier rang pour lequel Λ est de nouveau acceptable . La deuxième (S2) est la variante de la stratégie persévérante qui tend à augmenter initialement F' par un développement au début de l'algorithme de tous les sommets de H de rang inférieur ou égal à k (k = 2) ; S2 a été combinée à un backtracking long (stratégie S0).

Notons que l'incidence sur la complexité de l'algorithme du développement d'un sommet ou d'une actualisation est différente dans les trois stratégies S0, S1 et S2. Aussi, Y = n (D + 2.3 B) cesse d'être un indicateur valide de la complexité et il a fallu revenir à l'indicateur I (nombre d'opérateurs APL interprétés). La figure suivante fournit les courbes des valeurs moyennes de I en fonction de $\mathcal{E}$, pour n = 7, et sur les trois stratégies S0, S1 et S2.

Les valeurs correspondantes à $\mathcal{E} = \infty$ n'ont pas été indiquées, car elles sont identiques pour S0 et S1, la stratégie S2 ne présentant pas d'intérêt pour la recherche en profondeur sans backtracking.

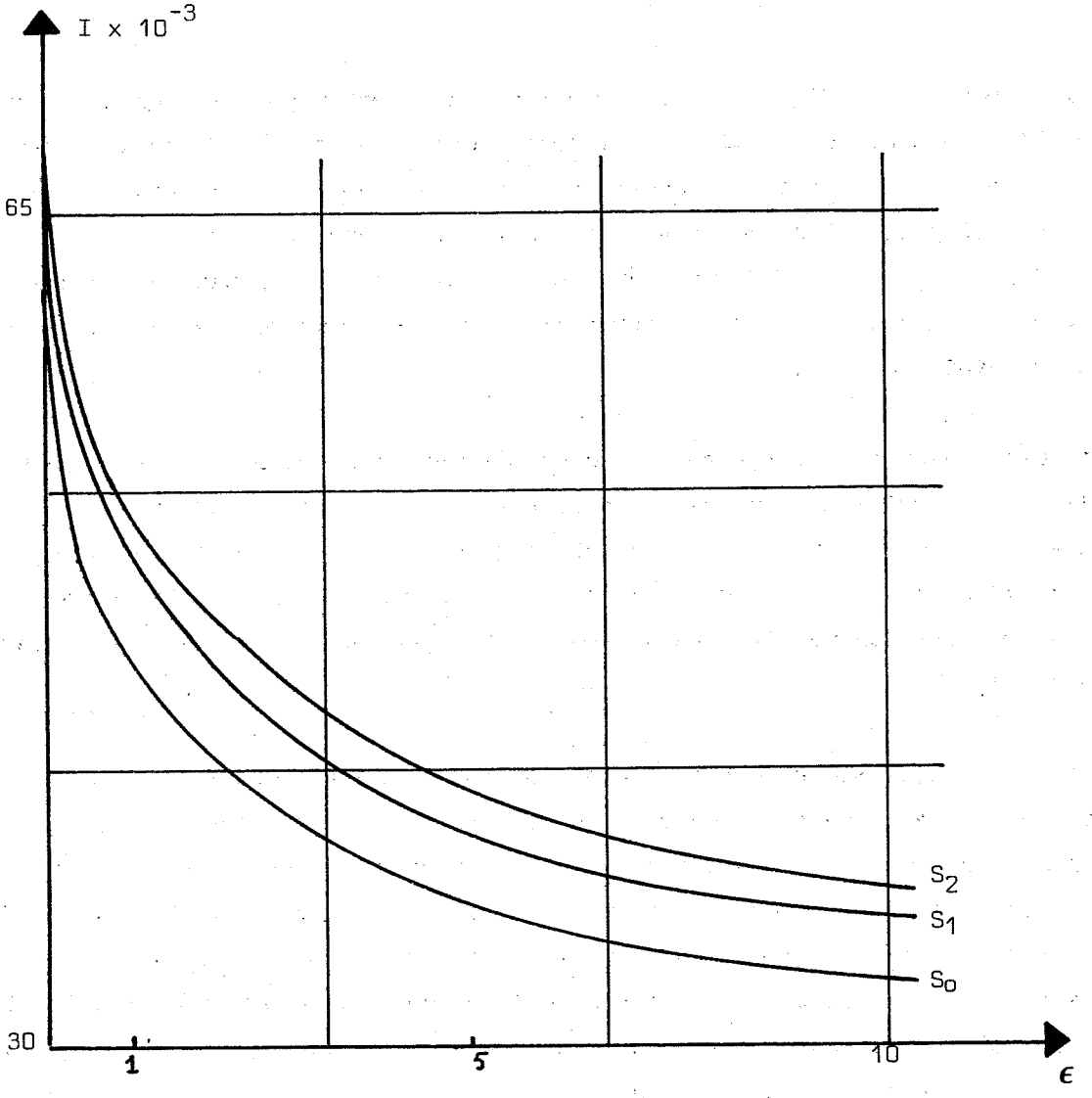


FIGURE 4.9.

On remarque que les trois courbes sont quasi-parallèles et conduisent à un ordre parfait : S0 meilleur que S1, qui est meilleur que S2. On prendra ce résultat avec quelques réserves, sachant que l'expérimentation a été restreinte à une seule valeur de n, et que les écarts types sur chacun des points de ces courbes se recouvrent largement. On retiendra cependant les conclusions qualitatives suivantes :

- l'inconvénient de S1 (nombre important de backtrackings et recherche tâtonnante en début d'exploration) se révèle primordial par rapport à son avantage (raccourcir la fin de l'exploration en concervant, autant que possible, l'acquis d'une solution presque complète et acceptable). On remarque, cependant, une légère tendance favorable pour $\varepsilon = 0$. Globalement, il semble que S1 n'est intéressante que dans le cadre d'une stratégie mixte qui démarre l'exploration avec S0 et la termine avec S1 ;

- le développement en largeur de H ne semble pas conduire à une valeur de seuil $(1+\varepsilon)y$ suffisamment élevée pour réduire la recherche au point d'amortir cet investissement initial. Il est probable qu'une stratégie telle que S2 ne devienne avantageuse que pour des valeurs plus élevées de n, et pour une variante un peu plus sophistiquée que le développement en largeur systématique pour l'élévation du seuil.

On terminera cette section par la question critique suivante : quel degré de confiance faut-il accorder aux modèles et aux diverses conclusions tirées de l'analyse des données enregistrées ? Autrement dit, dans quelle mesure ces données sont-elles typiques et représentatives de l'espace des données possibles ? Cela peut être estimé à un niveau qualitatif, si chaque mesure faite est un élément générique d'une classe de très faible variabilité, ou à un niveau quantitatif si l'expérimentation s'est basée sur un échantillon de dimension appréciable par rapport à celle de l'ensemble des données possibles.

Sur le plan qualitatif, on a déjà noté la variabilité considérable du problème : deux systèmes de règles identiques ne différant que par la distribution de coût peuvent conduire, pour la même valeur de ε , à des complexités de l'algorithme dans un rapport exponentiel.

Quantitativement, le nombre de systèmes de règles comportant a règles sur n caractéristiques binaires, éventuellement dépendantes, est exactement égal au nombre de partitions des 2^n sommets élémentaires du n-cube en (a + 1) blocs (les possibilités du générateur de données utilisé sont plus faibles, du fait de la restriction à Π_T notamment). En général, p éléments peuvent être partitionnés en q blocs de $\left\{ {p \atop q} \right\}$ façons possibles, $\left\{ {p \atop q} \right\}$ étant le nombre de Sterling de deuxième espèce :

$$\left\{ {p \atop q} \right\} = \sum_{0 \leq k_1 \leq k_2 \leq \ldots \leq k_{p-q} \leq q} k_1 \times k_2 \times \ldots \times k_{p-q} = \frac{(-1)^q}{q!} \sum_{k=0}^{p} \binom{p}{q} k^q (-1)^k$$

On en déduit que le nombre G(n) de systèmes de règles ayant au plus n caractéristiques binaires, avec ou sans dépendances, et a règles pour $2 \leq a \leq 3n$ est :

$$G(n) = \sum_{k=1}^{n} \sum_{j=2}^{3n+1} \left\{ {2^k \atop j} \right\} .$$

Calculé par la relation de récurrence $\left\{ {p \atop q} \right\} = q \left\{ {p-1 \atop q} \right\} + \left\{ {p-1 \atop q-1} \right\}$, en se limitant à n = 15, on trouve :

$$G(15) = 3.221 \times 10^{54427}.$$

(Une analogie peut aider à apprécier cet ordre de grandeur : pour se placer dans la situation équivalente à celle, peu confortable, d'un expérimentateur observant l'univers entier à partir d'un échantillon de 1 mg de matière, il nous faudrait enregistrer 10^{54367} points de mesures ; cela en admettant l'estimation actuelle de 10^{80} atomes dans l'univers).

Notons que l'ordre de grandeur de G(n) est suffisamment convaincant pour qu'il ne soit pas nécessaire de tenir compte davantage du nombre des distributions possibles des paramètres et des valeurs de ε.

La conclusion qui s'impose finalement est que les résultats de l'analyse empirique développée ne peuvent être pris comme modèle quantitatif précis de la complexité moyenne de l'algorithme qu'avec beaucoup de réserves.

Il faudra en retenir principalement les inforamtions qualitatives qui confirment et prolongent l'analyse faite en pire cas et certaines considérations intuitives (telle que la décroissance de l'ordre de complexité avec ε).

On peut conclure plus généralement que l'approche d'une modélisation empirique précise du comportement moyen d'un algorithme sur un problème combinatoire tel que le notre est, à priori, vouée à l'échec. Elle ne peut prendre un sens que dans le cadre d'une application qui restreint qualitativement et quantitativement l'espace des données possibles, le comportement de l'algorithme n'étant modélisé que pour cette application.

Un exemple type, dans notre domaine, est l'application à la Reconnaissance de Formes où l'on se donne, a priori, un ensemble de n caractéristiques, leurs relations de dépendance et leurs coûts. C'est en tant qu'illustration de ce qui pourraît être fait dans un tel cas qu'il faut lire l'ensemble de la section suivante.

°0°

4.4. APPROCHE POUR UNE OPIMISATION DYNAMIQUE

Un schéma d'approximation offre beaucoup plus de souplesse qu'un algorithme d'optimisation exacte ou qu'un algorithme approché. Son domaine d'application pratique est beaucoup plus étendu que le premier, et il fournit en plus du second et pour le même ordre de complexité, une garantie appréciable sur le degré d'approximation de sa solution, lequel degré peut être modulé en fonction des exigences et des moyens de calcul disponibles.

Un modèle de la complexité moyenne (si on lui accorde une confiance suffisante) accroît les avantages de l'algorithme et permet d'exploiter sa souplesse. Ayant une constante donnant le rapport entre l'unité de coût du PDF et l'unité de coût de son optimisation (coût de mise en oeuvre de AO_ε), et si on peut estimer le nombre total d'utilisation du PDF, on pourra répondre aux questions suivantes :

- est-il plus avantageux d'interpréter un système de règles ou de le compiler en arbre ?

- Dans ce dernier cas, avec quel degré d'optimisation ε ?

- Utilisant un PDF ε - optimal, si une modification intervient dans le système de règles, faudra-t-il le recompiler et avec quelle nouvelle valeur de ε ?

La dernière question généralise les deux précédentes et pose globalement le problème d'une optimisation incluant les deux critères : coût d'une solution - coût de sa recherche, et intégrant la dynamique du système.

Une approche à ce problème d'optimisation dynamique est développée dans cette section. Après avoir délimité le domaine d'application de l'algorithme AO_ε sur les PDF et analysé le choix du ε initial, on considèrera deux types d'évolutions d'un système de règles :

- les évolutions paramétriques : dérives ou meilleures estimations des distributions de coût et de probabilités, réévaluation du nombre total d'utilisations du PDF ;

- les évolutions structurales : ajout, suppression ou modification d'une règle, d'une relation de dépendance, d'une caractéristique.

4.4.1. DOMAINE D'APPLICATION DE L'ALGORITHME

Le domaine d'application de AO_ε peut être restreint :

- soit par une utilisation trop faible de l'arbre-solution ;
- soit par une dimension trop grande du système de règles S.

Le premier cas, correspondant à la borne inférieure du domaine, est assez faible à apprécier. Soit K une estimation du nombre total de processus décisionnels utilisant S (avec la même structure et les mêmes paramètres). Un modèle empirique du comportement de l'algorithme fournit une estimation :

- du coût $Y(n, \varepsilon)$ de mise en oeuvre de AO_ε sur S, exprimé dans les mêmes unités que le coût Ψ du PDF, et
- du coût relatif $\Psi_r\ (n, \infty)$.

Si S est compilé en un arbre t, au minimum le coût de compilation est $Y(n,\infty)$, et le coût des K processus décisionnels utilisant t est : $K \times \Psi_r\ (n,\infty) \times \sum_{j=1}^{n} p_j$.

Si S est interprété, le coût des K processus décisionnels est au plus $K \times \sum_{j=1}^{n} p_j$. D'où :

- Si $Y(n,\infty) > K \times (\sum p_j) \times (1 - \Psi_r\ (n,\infty))$, alors il est plus avantageux d'interpréter K fois le système S que de le compiler ;
- Sinon le coût $Y(n, \varepsilon)$ peut être amorti par l'utilisation d'un système compilé en arbre ε - optimal.

Quel degré d'approximation ε faut-il alors prendre ?

Le coût global d'utilisation d'un arbre $\hat{t}$ optimal est : $Y(n,0) + K \times \Psi(\hat{t})$.

Pour t_ε , arbre ε - optimal, ce coût est : $Y(n,\varepsilon) + K \times \Psi(t_\varepsilon)$.

On distinguera le cas particulier $\varepsilon = \infty$, où le coût de t_∞ est : $Y(n,\infty) + K \times \Psi(t_\infty)$.

En adoptant le modèle :

$Y(n,\varepsilon) = a(n) + b(n) \times c(n)^{-\varepsilon}$; et $Y(n,\infty) = a'(n)$; $a' < a \quad \forall n$

et en admettant que les majorants ε ou ε' sont proches des écarts réels :

$$\Psi(t_\varepsilon) = (1 + \varepsilon)\, \Psi(\hat{t})$$

$$\Psi(t_\infty) = (1 + \varepsilon'(n,\infty))\Psi(\hat{t})$$

avec l'estimation suivante de $\Psi(\hat{t}) = (\Sigma p_i) \times \Psi_r(n,0)$; on adoptera pour ε la valeur correspondante au minimum des trois expressions :

$$\begin{cases} a(n) + b(n) + K \times \Psi(\hat{t}) & (\text{pour } \varepsilon = 0) \\ a(n) + b(n) \times c(n)^{-\varepsilon} + K(1 + \varepsilon)\, \Psi(\hat{t}) & \\ a'(n) + K(1 + \varepsilon'(n,\infty))\, \Psi(\hat{t}) & (\text{pour } \varepsilon = \infty) \end{cases}$$

Cela revient à chercher le maximum de :

$$\begin{cases} 0 & (1) \\ b(1 - c^{-\varepsilon}) - K \times \varepsilon \times \Psi(\hat{t}) & (2) \\ b + (a - a') - K \times \varepsilon' \times \Psi(\hat{t}) & (3) \end{cases}$$

On vérifie facilement (Cf. Figure 4.10.) que le $\hat{\varepsilon}$ optimal est obtenu par les critères suivants :

- si $K > \max \left\{ b \log c / \Psi(\hat{t}) \; ; \; (b + a - a') / \varepsilon' \Psi(\hat{t}) \right\}$, alors $\hat{\varepsilon} = 0$,
  i.e., une compilation en un arbre optimal s'impose ;

- Sinon :

. Si $\varepsilon' > \frac{a - a'}{K\,\Psi(\hat{t})} + \frac{1}{\log c}\left[1 + \log\left(\frac{b \log c}{K\,\Psi(\hat{t})}\right)\right]$ alors

$\hat{\varepsilon} = \frac{1}{\log c} \times \log\left(\frac{b \log c}{K\,\Psi(\hat{t})}\right)$, valeur pour laquelle (2) est maximale et supérieure à (3) ;

. Sinon, $\hat{\varepsilon} = \infty$.

Remarquons que les écarts à l'optimum, basés dans ce qui précède sur des estimations pessimistes défavorables aux solutions ε - optimales, pourront être corrigés soit par une pondération globale (telle que $\Psi(t_\varepsilon) = (1 + \varepsilon/10) \times \Psi(\hat{t})$), soit plus finement si on dispose d'une estimation fiable de l'écart moyen réel à l'optimum.

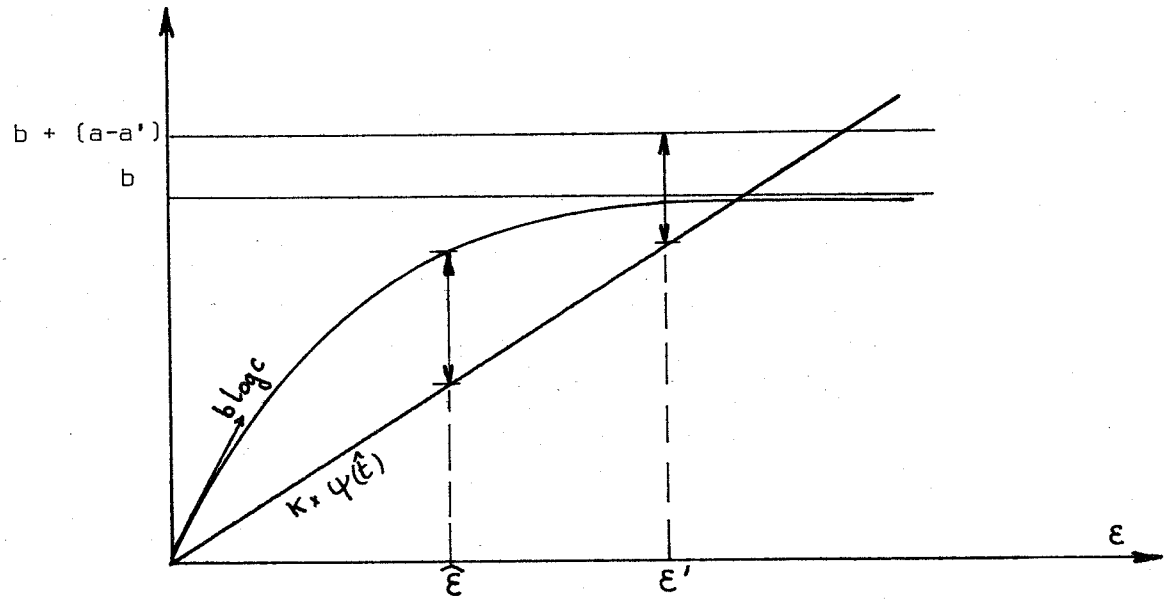


FIGURE 4.10.

Sachant que $Y(n, \varepsilon)$ croît exponentiellement en n, il n'est pas certain qu'une solution $\hat{\varepsilon}$ - optimale puisse être pratiquement accessible. On retrouve ici la borne supérieure du domaine d'application de l'algorithme AO_ε qui est déterminée par les ressources maximales de calcul dont on dispose.

Soit Ymax cette borne, traduite dans l'unité appropriée.

- Si $Y(n, \hat{\varepsilon}) > Ymax$ et $Y(n, \infty) \leqslant Ymax$, alors on se contentera d'un arbre ε - optimal pour $\hat{\varepsilon} < \varepsilon \leqslant \infty$ et $Y(n, \varepsilon) = Ymax$;

- Sinon, si $Y(n, \infty) > Ymax$, alors on procèdera à une compilation partielle avec $\varepsilon = \infty$, et limitée à n_o caractéristiques, pour n_o tel que :

  $Y(n_o, \infty) \leqslant Ymax$ et $Y(n_o+1, \infty) > Ymax$. Une alternative serait de générer un arbre (avec $\varepsilon = \infty$) par ordre de rang croissant, et cela jusqu'à épuisement des ressources ou achèvement de l'arbre.

Notons que la détermination de $\hat{\varepsilon}$ est inutile pour $n \geqslant n_o$.

4.4.2. EVOLUTIONS PARAMETRIQUES

Un aspect de la dynamique d'un système de règles S concerne les évolutions de ses distributions paramétriques de coût et de probabilité. Celles-ci peuvent varier dans le temps, du fait de réajustements périodiques de leurs estimations au fur et à mesure de l'utilisation du PDF, et éventuellement, du fait de dérivés propres.

Soit S un système appartenant au domaine d'application de AO_ε, et pour lequel un premier arbre t a déjà été obtenu. Un suivi de la dynamique de S consistera à :

- surveiller les évolutions des distributions de coût et de probabilité de S, et le nombre d'utilisations de t par le PDF ;

- réestimer le degré d'optimalité de t pour ces nouvelles distributions ; et

- si l'évolution est importante, ou en l'absence d'évolution si le nombre d'utilisation de t est plus grand que celui prévu, déterminer si une nouvelle compilation de S devient nécessaire et pour quelle valeur de ε .

Si t, de coût $\Psi(t)$, est ε - optimal dans les distributions (ρ, μ) ; il est simple de calculer son nouveau coût $\Psi'(t)$ dans les distributions (ρ', μ'), mais quel est son nouveau degré d'optimalité ε' ?

Soit $\hat{t}$ l'arbre optimal dans (ρ, μ), et $\hat{t}'$ l'optimum dans (ρ', μ'). Relativement au degré d'optimalité de t, dans le pire cas, la valeur du nouvel optimum a baissé : $\Psi(\hat{t}) \geqslant \Psi'(\hat{t}')$.

De plus, $\hat{t}'$ est sous-optimal dans (ρ, μ) : $\Psi(\hat{t}') \geqslant \Psi(\hat{t})$; d'où $\Psi(\hat{t}) - \Psi'(\hat{t}') \leqslant \Psi(\hat{t}') - \Psi'(\hat{t}')$.

Développons un majorant de l'écart $\Psi(s) - \Psi'(s)$ pour un arbre s quelconque dont le coût a baissé entre (ρ, μ) et (ρ', μ') :

$$\Psi(s) - \Psi'(s) = \sum_{\ell \in L} \sum_{x_i \in \upsilon(\Gamma^{-1}(\ell))} \left[\rho_i\, \mu(\ell) - \rho'_i\, \mu'(\ell)\right] \geqslant 0$$

$$= \sum_{\ell \in L} \sum_{x_i \in \upsilon(\Gamma^{-1}(\ell))} \rho_i \left(\mu(\ell) - \mu'(\ell)\right) \sum_{\ell \in L} \sum_{x_i \in \upsilon(\Gamma^{-1}(\ell))} \mu'(\ell)(\rho_i - \rho'_i)$$

$$= \sum_{\ell} \sum_{X_i} \rho'_i \left(\mu(\ell) - \mu'(\ell)\right) + \sum_{\ell} \sum_{X_i} \mu(\ell) \left(\rho_i - \rho'_i\right)$$

Dans les relations précédentes, certains termes sont positifs (pour $\rho_i > \rho'_i$ ou $\mu(\ell) > \mu'(\ell)$) et d'autres sont négatifs. On obtient un majorant en ne considérant que les termes positifs. Par ailleurs, l'ensemble des termes intervenant dans chacune de ces sommes (positifs et négatifs) est strictement inclus dans celui de tous les couples $\{l \in L\} \times \{1 \leqslant i \leqslant n\}$: on obtient un autre majorant en sommant sur tous les termes positifs de $\sum_{\ell \in L} \sum_{i=1}^{n}$; d'où :

$$\Psi(s) - \Psi'(s) \leqslant \sum_{\mu(\ell) > \mu'(\ell)} \sum_{i=1}^{n} \rho_i \left(\mu(\ell) - \mu'(\ell)\right) + \sum_{\ell} \sum_{\substack{i \\ \rho_i > \rho'_i}} \mu'(\ell) \left(\rho_i - \rho'_i\right)$$

$$\leqslant \sum_{i=1}^{n} \rho_i \times \sum_{\mu(\ell) - \mu'(\ell)} \left(\mu(\ell) - \mu'(\ell)\right) + \sum_{\substack{i \\ \rho_i > \rho'_i}} \left(\rho_i - \rho'_i\right)$$

du fait de $\sum_{\ell} \mu(\ell) = 1$. L'expression symétrique donne :

$$\Psi(s) - \Psi'(s) \leqslant \sum_{i=1}^{n} \rho'_i \times \sum_{\substack{\ell \\ \mu(\ell) > \mu'(\ell)}} \left(\mu(\ell) - \mu'(\ell)\right) + \sum_{\substack{i \\ \rho_i > \rho'_i}} \left(\rho_i - \rho'_i\right) \quad ; \text{ d'où :}$$

$$\Psi(s) - \Psi'(s) \leqslant \left[\sum_{i=1}^{n} \min\left\{\rho_i, \rho'_i\right\}\right] \times \left[\sum_{\mu(\ell) > \mu'(\ell)} \mu(\ell) - \mu'(\ell)\right] + \sum_{\rho_i > \rho'_i} \left(\rho_i - \rho'_i\right)$$

Posons : $\sigma_\rho = \sum_{i=1}^{n} \min \{ \rho_i , \rho'_i \}$

$\Delta\rho = \sum_{i} (\rho_i - \rho'_i)$ pour $1 \leq i \leq n$ et $\rho_i > \rho'_i$

$\Delta\mu = \sum_{\mu} (\mu(u) - \mu'(u))$ pour $u \in (\bigtimes_{i=1}^{n} Z_i)$ et $\mu(u) > \mu'(u)$

Sachant que $\mu(1) = \sum_{u \in 1} \mu(u)$; pour 1 tel que $\mu(1) > \mu'(1)$, on a :

$\mu(1) - \mu'(1) < \sum_{u} (\mu(u) - \mu'(u))$ pour $u \in 1$ et $\mu(u) > \mu'(u)$.

D'où, finalement, le majorant qui ne dépend que des distributions (ρ, μ) et $\rho', \mu')$:

$$\psi(s) - \psi'(s) \leq \sigma_\rho \, \Delta\mu + \Delta\rho \qquad (i)$$

En appliquant ce majorant à l'arbre $\hat{t}'$, on obtient :

$$\psi(\hat{t}) - \psi'(\hat{t}') \leq \psi(\hat{t}') - \psi'(\hat{t}') \leq \sigma_\rho \, \Delta\mu + \Delta\rho$$

résultat qui va être exploité pour majorer $\psi'(t)$ en fonction de $\psi'(\hat{t}')$:

$\psi'(t) = \Delta\psi + \psi(t)$; avec $\Delta\psi = \psi'(t) - \psi(t)$

$\leq \Delta\psi + (1+\varepsilon) \psi(\hat{t})$; car t est ε - optimal dans (ρ, μ)

$\leq \Delta\psi + (1+\varepsilon) [\psi'(\hat{t}') + \sigma\rho \, \Delta\mu + \Delta\rho]$; d'après (i)

$\leq \psi'(\hat{t}') \left[1 + \varepsilon + \dfrac{\Delta\psi + (1+\varepsilon)(\sigma_\rho \, \Delta\mu + \Delta\rho)}{\psi'(\hat{t}')} \right]$

Si t est ε'- optimal par rapport à $\hat{t}'$: $\psi'(t) \leq (1+\varepsilon') \cdot \psi'(\hat{t}')$.
D'où, en minorant le dénominateur et en identifiant :

$$\varepsilon' = \frac{\varepsilon + \alpha}{1 - \alpha} \qquad \text{avec} \quad \alpha = \frac{\Delta\psi + (1+\varepsilon)(\sigma_\rho \, \Delta\mu + \Delta\rho)}{\psi'(t)}$$

Pour des petites variations des paramètres, $\Delta\Psi$ pourraît être de l'ordre de $\Psi(t)/10$, $\Delta\mu$ et $\Delta\rho$ ne faisant intervenir que les écarts positifs, la somme ($\sigma_\rho \Delta\mu + \Delta\rho$) serait du même ordre de grandeur ; ce qui conduit à α de l'ordre de 0.1, et ε' assez proche de ε . Notons que α peut être négatif ($\Delta\Psi$ est calculé algébriquement) et inférieur à ε .

Une procédure de suivi de la dynamique du système, utilisant α comme indicateur de sa stationnarité, pourraît être la suivante :

A la suite d'une évolution des paramètres, on calcule $\Psi'(t)$, et les écarts par rapport à la distribution <u>initiale</u> (ρ, μ) : $\Delta\Psi$, $\Delta\rho$, $\Delta\mu$, σ_ρ

- Si la valeur de α correspondante est telle que $\alpha \leq \alpha_{lim}$: on estime que le système est resté stationnaire. On compare alors le nombre K' d'utilisations de l'arbre t déjà effectuées au nombre K qui était initialement estimé :

  . si $K' < K$: on poursuit sur l'utilisation du même arbre t ;

  . si $K' > K$: on réestime une nouvelle valeur K du nombre de processus décisionnels futurs utilisant t, compte-tenu de la stationnarité constatée. Sur cette base, on compare le coût de K utilisations de t par rapport au coût de K utilisations d'un arbre t' meilleur que t auquel s'ajouterait le coût d'une nouvelle compilation pour obtenir t'. Si un nouvel arbre s'avère avantageux, on détermine $\hat{\varepsilon} < \varepsilon$ par une procédure similaire à celle de la section précédente et on recherche un autre arbre $\hat{\varepsilon}$ - optimal, en poursuivant l'utilisation sur le meilleur de t et t'.

- Si $\alpha > \alpha_{lim}$: l'évolution est suffisamment importante pour détériorer le degré d'optimalité de t et remettre en question son utilisation. On réestime alors K et on reprend la procédure décisionnelle : poursuite de l'utilisation de t (dont le degré d'optimalité n'est plus que $\varepsilon' = (\varepsilon + \alpha)/(1 - \alpha)$ ou nouvelle compilation $\hat{\varepsilon}$ - optimale.

L'estimation de K doit prendre en compte la dynamique du système :

K doit être grand en cas de stationnarité et faible en période de grande variabilité.

On peut prendre heuristiquement $K \leftarrow \beta\ (\alpha_{lim} - \alpha)K$ pour une réestimation stationnaire, et $K \leftarrow K/\beta\ (\alpha - \alpha_{lim})$ dans le cas non stationnaire. Ainsi, pour une très longue période de stationnarité (système statique), K tendra vers l'infini et on convergera en quelques compilations vers un arbre optimal. Si, au contraire la dynamique est importante, K sera de plus en plus faible et le suivi de la dynamique se fera avec un écart à l'optimum d'autant plus grand que la variabilité est importante.

Notons finalement que α_{lim} pourra être pris de l'ordre de l'incertitude relative sur les distributions de coût et de probabilité ; et qu'éventuellement, on pourra corriger l'estimation $\varepsilon' = (\varepsilon + \alpha)/(1-\alpha)$ pour tenir compte de l'écart moyen entre le majorant ε et le degré réel d'optimalité ε_{π}.

4.4.3. EVOLUTIONS STRUCTURALES

Pour un grand nombre d'applications, l'utilisation du formalisme des systèmes de règles est avantageuse principalement à cause de sa souplesse. La littérature vantant cette qualité présente chaque règle de décision d'un système comme un "grain" de connaissance pratiquement indépendant ; ce qui permettrait au concepteur de se focaliser sur ce grain et de le modifier sans trop se préoccuper des couplages avec les autres règles. De plus, la même facilité pourrait être mise à profit par une procédure d'auto-modification qui, inférant par apprentissage une nouvelle règle, l'ajouterait au système.

Bien que l'indépendance des règles soit inexacte(si on veut conserver la consistance de l'ensemble), la souplesse du formalisme est effective, mais elle est malheureusement perdue par la compilation en arbre : si t représente S, toute modification dans les règles ou les caractéristiques de S annulera, en général, cette propriété. Dans quels cas la mise à jour de t est possible, et dans quels cas une recompilation s'impose ? A partir de quelle fréquence des évolutions structurales, l'approche compilation est à abandonner au profit d'une interprétation ?

Restreignons-nous aux évolutions structurales qui n'affectent que les règles d'un système (on examinera au chapitre 6 un cas particulier dans lequel l'ensemble des caractéristiques peut également être modifié). D'une manière générale, on peut ramener toute modification des règles à un ensemble d'ajouts et de suppressions de termes. Il est relativement simple de développer une procédure qui, à la suite d'une ou plusieurs de ces transformations élémentaires, permettrait de mettre à jour un arbre de décision de façon à ce qu'il continue à représenter le système de règles transformé. On pourrait, par exemple, procéder de la manière suivante :

- si u est un terme dont les sous-termes figuraient dans plusieurs règles $R_{j1}, \ldots, R_{j\alpha}$, et qu'il se trouve ajouté à la règle R_i, alors on parcourt l'ensemble des feuilles de t :

  . pour une feuille $l \subseteq u$: on modifie le label de l, en la faisant correspondre à R_i : $\mathcal{L}(l) \leftarrow A_i$;

  . pour une feuille l tel que $u \subset l$ et $\mathcal{L}(l) = A_i$: on remplace cette feuille par un sous-arbre représentant le sous-système S/l, car R/l contient plus d'une règle.

- si u est un terme qui figurait dans D et se trouve ajouté à R_i, alors on parcourt l'arbre t récursivement à partir de la racine :

  . pour un noeud v tel que $u \subset v$ et $\forall y \in \Gamma(v)$: $u \cap y = \emptyset$: on ajoute à v une branche supplémentaire menant à une feuille l avec $\mathcal{L}(l) \leftarrow A_i$;

  . pour v tel que $u \subset v$ et $\exists y \in \Gamma(v)$ avec $u \cap y \neq 0$: on reprend récursivement la procédure sur chaque $y \in \Gamma(v)$ avec $u \leftarrow u \cap y$ lorsque $u \neq 0$.

- si u est un terme qui figurait dans une ou plusieurs règles et qu'il se trouve ajouté à D, alors on parcourt les feuilles de t en élagant toutes celles incluses dans D, puis en élagant tous les noeuds qui n'ont plus qu'une seule branche.

La mise à jour d'un arbre de décision évite d'avoir à recompiler le système de règles à la suite d'une évolution structurale, mais on ne répond pas à la question initiale du point de vue de l'intérêt de cette mise à jour par rapport

à une recompilation. En effet, le coût de t mis à jour est immédiatement calculable, le coût de la mise à jour peut être estimé, mais les techniques de majorations de la section précédente ne permettront pas d'évaluer le nouveau degré d'optimalité de t. Une modification, même minime, dans la répartition des termes de S peut avoir un impact considérable sur le coût du nouvel optimum. Ni le minorant dont on dispose à priori ($g(u_o) = \sum_{j=1}^{n} p_j q_j$), ni la variation du coût de t ne sont d'une quelconque utilité pratique pour déterminer ε'.

En l'absence d'un critère permettant de décider de l'opportunité d'une mise à jour ou d'une recompilation (critère qui restera parmi les nombreux problèmes ouverts dans cette approche à une optimisation dynamique), on peut procéder empiriquement comme suit :

- on fixe un maximum de r mises à jour successives sur le même arbre, au-delà desquelles une recompilation s'impose. Le coût du nouvel arbre permettra de situer la dégradation de l'arbre mis à jour, et éventuellement, de modifier r en conséquence ;

- si $K < \frac{1}{r} \frac{Y + rJ}{(1 - \Psi_r) \sum_{j=1}^{n} p_j}$, avec J : coût moyen d'une mise à jour, et

  K : estimation du nombre de processus décisionnels sur S entre deux évolutions structurales, alors il est plus intéressant d'interpréter le système de règles que de le compiler.

°0°

4.5. CONCLUSION

Ce chapitre, qui termine la 1ère partie du mémoire, a été consacré à l'étude de la complexité du problème d'optimisation des PDF.

La caractérisation théorique de cette complexité révèle deux aspects :

(i) la non-appartenance du problème décisionnel correspondant à la classe NP, et

(ii) l'inexistence de schéma d'approximation polynomial, ou d'algorithme à approximation garantie polynomial (à moins que P=NP), et cela même pour les instances où le problème décisionnel est dans NP.

Le premier point explique le comportement des algorithmes $A1(\gamma)$ à approximation non bornée (Chapitre 1), et permet d'argumenter en faveur d'un schéma d'approximation tel que AO_ε, pour les dimensions des instances rencontrées en pratique, et en particulier pour celles des applications traitées en deuxième partie du mémoire.

Le modèle empirique de la complexité moyenne d'AO_ε qui a été développé par la suite renforce cette argumentation et illustre l'intérêt de l'approche ε-admissible par rapport à celle admissible.

La deuxième section s'est attachée à délimiter le domaine d'utilisation de l'algorithme proposé, et à développer des procédures de choix du paramètre ε en fonction du nombre d'utilisations d'un PDF et du modèle de comportement de l'algorithme.

DEUXIEME PARTIE

QUELQUES PROCESSUS DECISIONNELS EN ROBOTIQUE

CHAPITRE 5

UN SYSTEME D'IDENTIFICATION D'OBJETS

5.1. INTRODUCTION : PRESENTATION DE L'APPLICATION

5.2. IDENTIFICATION D'OBJETS COMPLETEMENT OBSERVES

5.2.1. La Reconnaissance de Formes Séquentielle : approches connues et leurs difficultés

5.2.2. Partitionnement de l'espace de représentation

5.2.3. Génération d'un système de règles d'identification

5.2.4. Génération du classifieur

5.2.5. Mise en oeuvre expérimentale

5.3. APPROCHE A UNE IDENTIFICATION SYNTAXIQUE D'OBJETS PARTIELLEMENT OBSERVES

5.3.1. Modèle et représentation d'objets plans

5.3.2. Apprentissage

5.3.3. Identification

5.3.4. Algorithmes de recherche de facteurs dans un mot

5.4. CONCLUSION

°0°

5.1. INTRODUCTION : PRESENTATION DE L'APPLICATION

Il n'est pas nécessaire, pour souligner l'importance de la perception en robotique, de revenir à de longues généralités sur cette discipline, ni d'avoir à redéfinir les quatre ou cinq fonctions de base indispensables dans un robot de la $i^{ème}$ génération (voir par exemple G. GIRALT (74)) pour qui : "C'est l'existence de la fonction de perception de l'environnement...qui sera un des aspects dominants de la seconde génération de robots, les seuls justifiant réellement de ce nom aux yeux des scientifiques, etc...").

De très nombreux problèmes de perception se posent en robotique, parmi lesquels on pourrait citer la liste (non exhaustive) suivante :

- les problèmes de localisation-positionnement : présence/absence d'un objet, localisation simple le long d'un axe, localisation dans le plan, localisation dans l'espace, positionnement relatif de plusieurs objets en mouvement, etc... ;

- les problèmes de classification : tri d'objets isolés, identification dans un vrac planaire (peu de recouvrements), identification dans un vrac volumique, objets plans ou tridimensionnels fixes ou en mouvement, etc... ;

- les problèmes d'inspection : macro-inspection, analyse de conformité, interprétation de tolérances, micro-inspection et traitements fins d'états de surface, etc... ;

- les problèmes d'analyse de scènes.

Le contexte spécifique de la robotique ajoute à ces problèmes de nombreuses contraintes et les infléchit dans une formulation souvent assez différente de celle qu'ils peuvent avoir dans d'autres applications de la reconnaissance de formes. Aussi, la plupart de ces problèmes n'ont pas encore trouvé de solution entièrement satisfaisante et posent, à des degrés de difficultés variables, des questions non résolues au niveau :

- des capteurs utilisés ;
- des systèmes de numérisation, filtrage, et prétraitement du signal physique ;
- des algorithmes d'interprétation et de traitement de l'information sensorielle ;
- des méthodes d'apprentissage (ou d'acquisition rapide en conversationnel) des algorithmes précédents ;
- des modèles de représentation d'objets et des méthodes d'acquisition de ces modèles ;
- des structures informatiques particulières de systèmes de perception, etc...

Parmi toutes ces questions, ce chapitre n'aborde qu'un problème très restreint et sous l'angle relativement étroit des processus décisionnels. Il s'agit du problème d'identification d'objets plans.

Le qualificatif plan recouvre ici des notions relatives à la fois aux objets perçus et au type de perception qui en est faite. Plus précisément, on admet en particulier les hypothèses suivantes :

(H1) les objets considérés possèdent une, ou un nombre très faible de positions d'équilibre stable sur le plan horizontal (par exemple, objets plats dont une dimension est négligeable par rapport aux autres) ;

(H2) les diverses classes d'objets à identifier sont reconnaissables à leur "ombre", i.e. à la projection sur le plan horizontal de leur contour externe (et interne s'ils ont des trous) ;

(H3) les objets perçus sont isolés les uns des autres, ou bien sont en vrac planaire avec un nombre très faible de niveaux de superposition, et tel que ces superpositions n'introduisent pas de trop grandes déformations de perspectives susceptibles d'annuler (H2).

Dans une application typique de ce problème d'identification d'objets plans, une caméra vidéo surplombe un plan de travail fixe ou un tapis roulant.

Diverses pièces mécaniques plates sont posées sur le plan de travail, ou bien arrivent sur le tapis roulant qui en a été alimenté par des gouttières à un débit assez lent pour garantir (H3). Le champ de la caméra est de l'ordre de quelques unités de la dimension (plane) la plus grande des objets à percevoir, et les dimensions de tous ces objets ont des ordres de grandeur similaires. La lumière arrive à la caméra par éclairage direct (plan de travail lumineux), ou par réflexion si l'état de surface et la couleur des objets donnent une image suffisamment constrastée pour garantir (H2). Un ou plusieurs bras manipulateurs opérant sur le même plan de travail (ou en aval sur le tapis roulant) sont chargés de la prise de ces objets en vue, par exemple, de réaliser un assemblage, de charger une machine-outil, et/ou tout simplement, de les ranger dans l'emballage qui leur correspond.

Dans cette application, la tâche de la perception consiste donc à :

1) localiser les objets avec une précision suffisante pour permettre leur prise ; ce qui correspond à la détermination de leur face d'équilibre, de leur barycentre, et de leur orientation dans le plan ;

2) identifier les objets perçus parmi l'ensemble des classes connues, ou bien comme étant des objets inconnus ;

3) éventuellement, effectuer une macro-inspection des objets reconnus (par exemple bord cassé, trou manquant, etc...).

Trois contraintes essentielles de fiabilité, rapidité et coût doivent être prises en compte :

- Fiabilité : très faible taux d'erreur d'identification (inférieur à 1 %). Par contre, on admet un taux de non identification plus grand (de l'ordre de quelques pour-cent). En effet, une identification incorrecte entraînera une manipulation erronée de l'objet avec, éventuellement, une incidence très coûteuse sur la suite du processus (blocage d'une chaîne d'assemblage, détérioration d'un outil d'usinage, etc...), alors que cet objet classé comme inconnu sera recyclé devant le poste de perception et trié dans une classe particulière si un deuxième échec a lieu.

- Rapidité : le temps de traitement complet d'une image contenant 1 à 5 objets doit être de l'ordre de la seconde (éventuellement, moins dans le cas du tapis roulant).

- Coût : cette contrainte intervient au niveau du matériel mis en oeuvre et de son système de programmation :

  . le système de perception ne doit pas immobiliser, en temps réel, des moyens de calculs considérables. On ne disposera généralement que d'un ensemble autonome de trois modules :

    1) Caméra ;

    2) Sous-système spécialisé de numérisation et prétraitement de l'image, et

    3) Micro-processeur réalisant l'identification proprement dite.

  . l'acquisition du modèle des objets à identifier et de leur programme d'identification ne doit pas nécessiter une longue programmation. Une période de l'ordre de l'heure devrait suffire à un opérateur utilisant, en conversationnel, un logiciel d'apprentissage (sur, éventuellement, des moyens de calcul plus puissants), en plaçant successivement dans le champ de la caméra et sous diverses conditions de positions, d'éclairage et de face d'équilibre, plusieurs échantillons de la même classe à apprendre.

La satisfaction de ces deux dernières contraintes de rapidité et de coût est facilitée, dans notre cas, par l'hypothèse (H2). Elle permet, en effet, d'effectuer une discrétisation de l'image à deux niveaux, noir et blanc, et de restreindre toute l'information utile à un ensemble de contours fermés et à leur arbre d'inclusion.

Pour donner une idée de ce prétraitement, tel qu'il a été réalisé dans l'application développée au L.A.A.S., les figures suivantes (Figure 5.1.) présentent :

(a) une photographie des objets à percevoir ;

(b) l'image discrétisée en noir et blanc ;

(c) les contours extraits ; et

(d) leur arbre d'inclusion.

Le passage de l'image (a) à la mise en mémoire des listes complètes codant l'information (c) + (d) est effectué en 20 ms par un matériel de prétraitement spécialisé (vitesse de balayage du signal vidéo). Ce système fournit, en plus, les barycentres des contours extraits et, éventuellement, d'autres caractéristiques. On ne donnera pas ici de détail supplémentaire sur les problèmes inhérents à ce prétraitement (seuil de discrétisation, codage des points, etc...), on renvoie au travail de Bourdeau et al (17,231) , Stuck (231) et Talou (232) .

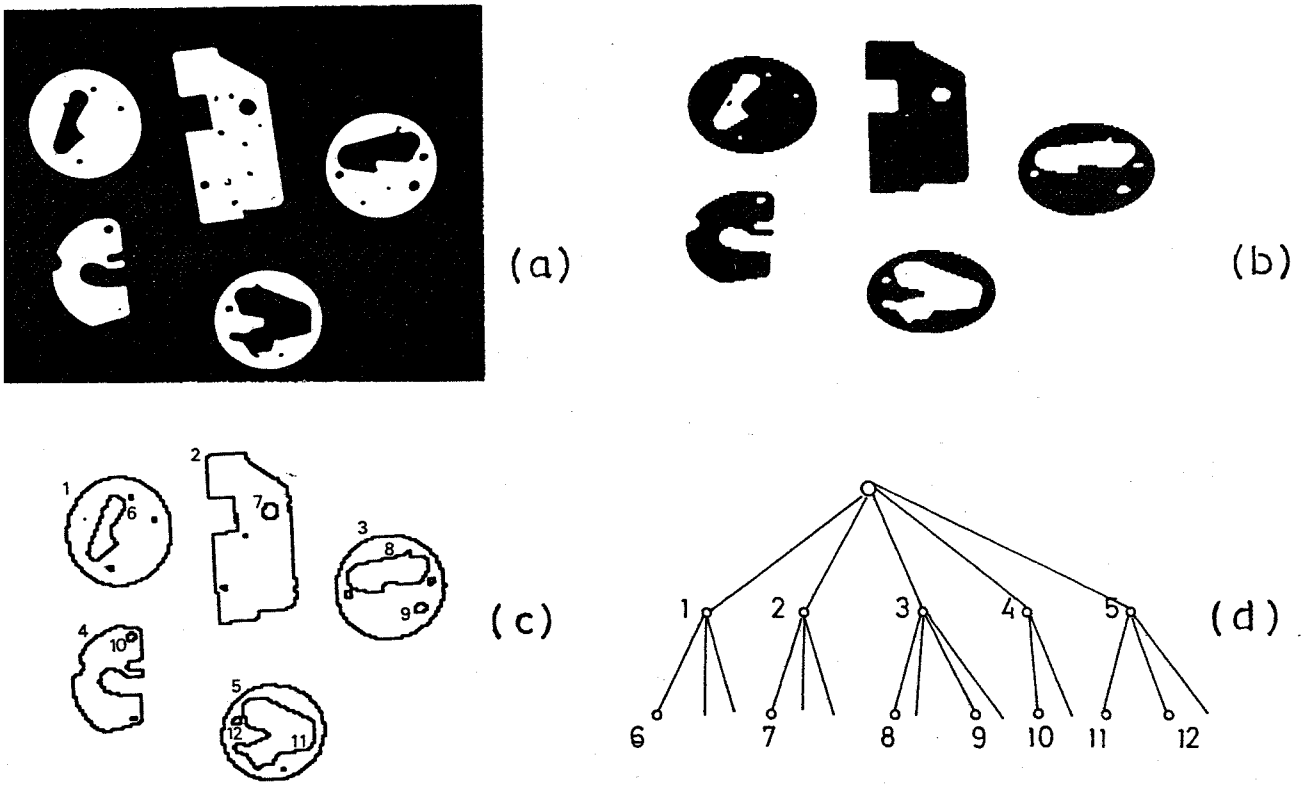


FIGURE 5.1.

Les contours extérieurs (de rang 1 dans l'arbre d'inclusion) sont alors successivement traités. On tente d'interpréter un contour ne se refermant pas sur le bord de l'image comme un objet isolé entièrement perçu. Si on arrive à

l'identifier (sur la base de ses caractéristiques globales), on confirme l'identification par quelques tests supplémentaires destinés également à achever une localisation précise (en orientation et symétrie miroirs par exemple).

Si l'identification échoue (non identification ou non confirmation), ou bien pour un contour partiellement hors du champ de vision, on tente une identification sur la base de caractéristiques locales et par des méthodes syntaxiques du ou des contours en recouvrement perçus.

Ainsi, on peut distinguer deux grandes classes de problèmes d'identification d'objets plans :

- identification d'objets isolés entièrement observés ;

- identification d'objets en recouvrement et/ou partiellement observés.

Ce chapitre présente une méthode originale assez complète pour la résolution du premier problème dans le contexte des systèmes de reconnaissance de formes séquentielles. La contribution porte sur l'aspect processus décisionnels d'identification : partitionnement des caractéristiques, génération d'un ensemble de règles de décision (d'identification) fiables et leur compilation en un arbre ε - optimal.

La mise en oeuvre expérimentale de la méthode proposée a fait l'objet de la thèse de F. Stuck (231) . On renvoie à ce document pour tous les aspects intégration système et pour la résolution des problèmes de localisation et de vérification. L'exposé sera donc restreint à la contribution méthodologique. Après une rapide présentation de la Reconnaissance de Forme Séquentielle (RFS), de quelques approches connues dans la littérature et de leurs difficultés, on introduira les diverses étapes de la méthode proposée en mettant l'accent sur les aspects spécifiques qui s'écartent du modèle général des PDF présenté précédemment.

La troisième et dernière partie du chapitre est une approche au problème d'identification syntaxique d'objets en recouvrement.

On présentera brièvement les grandes lignes de la problématique de recherche sur laquelle le travail se poursuit, puis on développera quelques algorithmes originaux de recherche de facteurs dans un mot (string-matching) qui, indépendamment de leur intérêt propre, peuvent être avantageusement utilisés dans cette application.

°0°

5.2. IDENTIFICATION D'OBJETS COMPLETEMENT OBSERVES

5.2.1. LA RECONNAISSANCE DE FORMES SEQUENTIELLES (RFS). APPROCHES CONNUES DIFFICULTES

En essayant de respecter les notations habituelles en Reconnaissance de Formes (pour un ouvrage de référence, voir par exemple (34,49)), on définira :

- $\mathcal{E}$: ensemble des ensembles de données d'entrées possibles. Ces données sont fournies par le prétraitement du signal physique que génère le capteur. On suppose, dans cette section, que le prétraitement réalise aussi la segmentation de l'information initiale (par exemple, segmentation de l'image), de telle sorte que les données e à interpréter ne concernent qu'un objet unique. Dans le cas de l'application citée, e constituera l'ensemble des points codant un contour externe et tous ses contours internes (sous-arbre de rang 1 dans la figure 5.1.c). On dira que e est l'objet inconnu à identifier.

- $\Omega = \{\omega_1, \ldots, \omega_m\}$: ensemble des classes d'objets à un élément duquel le système de perception doit identifier e. On écrira $e \in \omega_j$ si les données e sont reconnues comme étant celles correspondantes à un objet de la classe ω_j.

- $X = \{X_1, \ldots X_n\}$: ensemble de fonctions caractéristiques, chaque X_i étant une application partielle de $\mathcal{E}$ sur un domaine K_i ayant une bonne structure mathématique (en général, le corps $\mathbb{R}$).

Exemple 5.1. : L'application développée au L.A.A.S. utilise, entre autres, les caractéristiques suivantes :

- surface d'un contour ;
- finesse (carré du périmètre divisé par la surface) ;
- distance moyenne centre de gravité - point du contour ;

- rapport de la distance minimale (ou maximale) à la distance moyenne ;

- nombre de trous ;

- rapport de la surface totale des trous à celle du contour externe.

On fait habituellement référence à Ω comme étant l'espace de décision du système et à $(\bigtimes_i K_i)$ domaine d'application des caractéristiques comme étant son espace de représentation. Le passage de $(\bigtimes_i K_i)$ à Ω n'est pas déterministe et leur relation est souvent mal connue et modélisée de façon approximative.

Dans le cas de la représentation paramétrique, on suppose que les valeurs possibles de X_i pour tous les objets de la classe ωj sont celles d'une variable aléatoire distribuée suivant une loi de probabilité connue. On estime les paramètres de cette loi sur quelques échantillons d'objets de ω_j. Si les caractéristiques sont conditionnellement indépendantes, on modélisera chaque X_i par l'ensemble des n densités de probabilités de X_i sachant ω_j :
$P_{ij} : K_{ij} \dashrightarrow [0,1]$.

Cette représentation commode n'est pas toujours possible (par exemple, caractéristiques binaires, distributions ne ressemblant à aucune loi simple, dépendances exigeant l'estimation de densités multi-dimensionnelles dans $\bigtimes_i K_i$). Aussi, dans le cas de la représentation non paramétrique, on acquiert initialement et on retient un nuage de points $(X_1(e), \ldots, X_n(e))$, chaque point étant étiqueté par la classe ω_j à laquelle appartient l'objet correspondant à l'échantillon e. Dans certains cas, ce nuage peut être réduit par des techniques statistiques classiques, par exemple partitionnement d'un axe K_i, comptage de fréquences et construction d'un histogramme par classe d'objets.

En RFS, l'identification s'effectue en évaluant successivement sur e certaines caractéristiques de X. A chaque étape, on exploite l'information obtenue, $X_i(e)$, pour soit poursuivre le processus par l'évaluation d'une autre caractéristique, soit l'arrêter si on dispose de suffisamment d'informations pour classer e. Le principal problème, celui de l'apprentissage, est de définir à partir d'un ensemble d'échantillons étiquetés un algorithme appelé le classifieur qui précisera chacune des étapes du processus décisionnel précédent.

Ce problème sous-tend plusieurs sous-problèmes (non indépendants),dont les principaux portent sur :

(i) la définition des caractéristiques : il s'agit de définir ou de compléter automatiquement l'ensemble X. On cherche quelles pourraient être les transformations les plus discriminantes à appliquer aux données e, compatibles avec les échantillons étiquetés d'apprentissage et permettant l'identification de e ;

(ii) la partition de l'espace de représentation : on voudrait savoir où placer dans $(\bigtimes_i K_i)$ les surfaces de séparation entre classes d'objets, de façon à ce que chaque point corresponde à une classe unique et que cette correspondance soit compatible avec les échantillons d'apprentissage ;

(iii) la sélection des caractéristiques pertinentes : on dispose, en général, d'un ensemble X très redondant, on en cherche un sous-ensemble qui soit discriminant pour les classes d'objets considérées ;

(iv) la génération du classifieur : il s'agit de définir dans quel ordre (séquence linéaire ou structure plus complexe) évaluer les caractéristiques et à quoi comparer chaque valeur obtenue pour déterminer dans quel sous-domaine de $(\bigtimes_i K_i)$ se trouve le point $(X_1(e), \ldots X_n(e))$.

Remarque : Certains auteurs (108,131) distinguent les classifieurs séquentiels des classifieurs hiérarchisés ; ils restreignent les premiers à un ordre total sur les caractéristiques et admettent, pour les seconds, une structure arborescente. L'aspect séquentiel étant présent dans les deux cas, on ne suivra pas cette distinction ici.

Ces problèmes sont résolus en optimisant comme critère de coût général la somme pondérée de deux composantes :

- le coût d'identification (correspond souvent à un temps moyen) ;
- le risque d'identification (espérance de coût de l'erreur).

On peut éventuellement tenir compte de l'espérance de coût de non identification et du coût de l'apprentissage (taille des échantillons, complexité des algorithmes, temps opérateur s'ils sont interactifs, etc...).

Peu de travaux traitant du problème (i) me sont connus. L'approche de Roche (206)est criticable dans le sens où elle se restreint initialement à un jeu d'opérateurs trop élémentaires et qu'elle cherche à générer une caractéristique unique entièrement discriminante pour l'application considérée.

Le plus souvent, on est au contraire en mesure de définir, à priori, une collection de fonctions caractéristiques couvrant une large classe d'applications et le problème (i) ne se pose pas. C'est ce que nous supposerons, en admettant que X est une donnée pour les problèmes (ii) à (iv).

L'approche Bayesienne est une des méthodes de classification les plus connues et dont il n'est pas nécessaire de rappeler la formulation (Cf. par exemple (34,49)). Ses principaux avantages sont d'éviter le problème (ii) des surfaces de séparation et de minimiser la probabilité d'erreur. Par contre, elle possède de nombreux inconvénients, parmi lesquels la lourdeur des calculs mis en oeuvre (actualisation des probabilités à postériori à chaque étape), la difficulté à prendre en compte des caractéristiques structurales (relation d'inclusion, par exemple), et la nécessité de disposer de densités multidimensionnelles ou d'admettre l'hypothèse d'indépendance conditionnelle des caractéristiques.

De plus, on ne sait pas bien résoudre les problèmes (iii) et (iv) dans l'approche Bayesienne : la seule formulation connue est celle de Fu (48) restreinte à ses caractéristiques conditionnellement indépendantes. Pour un ordre total d'évaluation des caractéristiques donné à priori, son approche utilise la programmation dynamique pour sélectionner un sous-ensemble de caractéristiques pertinentes. Inversement, on peut à partir d'un sous-ensemble de caractéristiques, donné à priori, déterminer un ordre linéaire de leur évaluation optimisant un critère coût (temps de classification - risque d'erreur).

De nombreuses autres approches d'identification statistiques existent, mais aucune, à ma connaissance, ne résoud les problèmes mentionnés dans l'optique de la RFS.

Comme plusieurs auteurs l'ont remarqué, l'aspect séquentiel apparente ces problèmes à celui du tri (informatique) où l'outil arbre est particulièrement adapté.

La méthode de Agin et Duda (1) l'utilise d'une manière assez rudimentaire. Elle modélise toutes les caractéristiques supposées indépendantes, par des densités Gaussiennes de même écart type. Pour chaque caractéristique, on calcule les distances entre moyennes consécutives des différentes classes et on place un seuil au milieu de l'écart le plus grand. La caractéristique X_i dont cet écart est maximal, est choisie pour racine d'un arbre binaire qui partage le problème entre classes dont la moyenne est inférieure au seuil et celles dont la moyenne est supérieure. Le processus se répète sur chaque sous-ensemble de classes (jusqu'à réduction à un singleton), et avec toutes les caractéristiques de départ. Ainsi, une même caractéristique peut se retrouver en plus d'un noeud d'un chemin de la racine à une feuille.

L'avantage de cette méthode est l'extrême simplicité de sa mise en oeuvre. Ceci a permis d'ailleurs de l'intégrer, avec quelques extensions, à un système de vision développé industriellement (Machine Vision, Automatics). Ses inconvénients sont la restrictivité des hypothèses initiales, la non prise en compte d'un quelconque critère de coût d'identification, mais surtout l'absence de tout résultat sur la probabilité d'erreur du classifieur obtenu (aucune donnée expérimentale sur les performances des modules commercialement disponibles ne m'est connue).

Slagle et Lee (226) s'appuient sur l'inférence Bayesienne et ils apparentent la classification, non pas au problème du tri, mais à un jeu contre la nature. Ils construisent un arbre de jeu où à chaque état (noeud de décision), on a le choix entre :

- quitter le jeu : identification à la classe la plus probable avec risque d'erreur ; et

- poursuivre le jeu en évaluant une des caractéristiques restantes (compromis coût d'évaluation - information apportée).

L'arbre est construit avec une méthode $\alpha - \beta$ à profondeur variable (procédure γ). Le principal inconvénient de la méthode est d'élaborer un arbre qui ne peut servir qu'une seule fois et qui est reconstruit à chaque processus d'identification (à chaque partie du jeu).

L'approche de Stoffel (229) est différente. Constatant l'échec des méthodes statistiques sur des caractéristiques structurales (ou ce qu'il appelle caractéristiques discrètes de type 2, i.e. dont le domaine est de très faible cardinalité), il propose de réduire le nuage de points des échantillons d'apprentissage par une méthode d'implicants premiers. On modélise l'ensemble des points d'une même classe par un nombre réduit d'implicants sans recouvrement avec ceux des autres classes. L'identification se fait moyennant une distance de Hamming.

On peut étendre sa méthode en compilant les implicants obtenus en un arbre de décision. C'est ce que font Sethi et Chatterjee (220) dans un cas particulier (caractéristiques binaires et approche des questionnaires : tout implicant non explicité est impossible), et avec une heuristique assez complexe à calculer qu'ils ne justifient pas.

Les contributions aux problèmes (iii) et (iv) de plusieurs auteurs (par exemple Meisel (157) , Payne (176) , Dattatreya (28)) peuvent se ramener, à quelques variantes près, à l'approche suivante : compte-tenu des échantillons d'apprentissage, on partitionne l'espace de représentation par des hyperplans orthogonaux (suivant des techniques classiques de clustering). On formule ensuite, d'une manière récurrente, un critère somme du coût d'identification et du coût d'erreur, critère que l'on optimise par la programmation dynamique en construisant un arbre de test binaire dont chaque noeud est la comparaison de la valeur d'une caractéristique mesurée à un hyperplan frontière. Sur un même chemin de l'arbre, la même caractéristique peut être rencontrée plusieurs fois avec, à chaque fois, le même coût (celui d'une comparaison).

L'approche de Kulkarni et Kanal (109,131) est plus ambitieuse que les précédentes. Ils ne se restreignent, dans leur formulation initiale, à aucun type particulier de caractéristique. Ils posent globalement, sans les décomposer, les problèmes (ii), (iii), et (iv), et ils prennent en compte dans le critère à la

fois le coût d'identification et le risque d'erreur. Le premier modèle qu'ils adoptent est celui d'une recherche heuristique dans un espace d'état de dimension $2^n 2^m$ (chaque état étant défini par un sous-ensemble de caractéristiques mesurées et par un sous-ensemble de classes restant possibles), mais alors l'identification de chaque objet nécessite la mise en oeuvre d'un algorithme A^* de complexité exponentielle (en n et m). Devant l'inefficacité du classifieur obtenu, ils se proposent de restreindre le graphe d'état à un de ses sous-arbres, en choisissant parmi les sous-arbres possibles celui optimal en moyenne. La formulation de ce choix par Programmation Dynamique, même en se restreignant à des caractéristiques indépendantes et des arbres binaires et où chaque classe figurera une et une seule fois dans une feuille (questionnaire), leur apparaît très vite comme infaisable.

Contraints de faire des hypothèses restrictives supplémentaires, Kulkarni et Kanal adoptent l'optique d'un système interactif où l'opérateur fournirait le "squelette" de l'arbre d'identification (i.e. un arbre binaire ayant m feuilles étiquetées d'une façon biunivoque par les classes de Ω), le système se chargeant (par Branch-and-Bound) de déterminer les caractéristiques à mettre aux noeuds de cet arbre. On voit mal comment l'opérateur pourrait fournir au système une structure aussi élaborée à partir de considérations intuitives et des informations partielles dont il peut disposer à priori, et l'argumentation des auteurs cités est peu convaincante sur ce point.

De très nombreux autres travaux sur la RFS ont été publiés dans la littérature, et ce qui précède est loin de constituer un exposé complet sur les diverses méthodes connues (cf.(2,10,24,50,107,225)). Les références citées ont été restreintes à celles qui permettraient au lecteur de situer la contribution de l'approche développée ici.

Les grandes lignes de cette méthode reposent sur les choix simplificateurs suivants :

1) Il n'est pas tenu compte de l'erreur d'identification comme composante du critère à optimiser. On la considère comme une contrainte à respecter : la probabilité d'erreur du classifieur généré devra être inférieure à η ,paramètre fourni à priori ;

2) On n'exige pas du classifieur d'identifier tout ensemble de données et on admet une probabilité de non classification sensiblement plus élevée que la probabilité d'erreur ;

3) Grâce à (1), on peut décomposer le problème en : (ii) partitionnement de l'espace de représentation vérifiant la contrainte sur la probabilité d'erreur, et (iii) plus (iv) sélection des caractéristiques et leur organisation en un arbre de décision ;

4) Le problème (ii) de partitionnement de l'espace de représentation est résolu en introduisant des zones de non classification. Seules les zones étiquetées doivent satisfaire la contrainte sur la probabilité d'erreur.

5.2.2. PARTITIONNEMENT DE L'ESPACE DE REPRESENTATION

On admet, à ce niveau, l'hypothèse restrictive d'indépendance conditionnelle des caractéristiques, hypothèse dont on envisagera la relaxation ultérieurement.

A partir de là, le problème du partitionnement de l'espace de représentation ($\bigtimes_i K_i$) se ramène à celui du partitionnement de chacun des axes K_i.

L'approche suivante reste valable en représentation paramétrique ou non paramétrique et, quel que soit le type de caractéristiques utilisées. Pour des raisons de simplicité, on la développera initialement sur le cas paramétrique.

L'argumentation repose sur les trois propositions suivantes :

Proposition 5.1. : Le partitionnement par seuil unique entre deux classes ne permet pas, en général, de respecter une contrainte sur la probabilité d'erreur.

Proposition 5.2. : L'introduction d'une bi-partition par classe (sous-domaine de sélection et sous-domaine d'exclusion pour chaque classe ω_j) permettra de respecter cette contrainte, mais ne garantira pas une identification de l'objet inconnu à la classe la plus probable (compte-tenu de l'information disponible).

Proposition 5.3. : Le partitionnement de chaque axe K_i par produit de tri-partitions, une pour chaque classe (sélection, exclusion, non-classification) permet une identification identique à celle que fournirait un classifieur Bayesien, et donc avec une probabilité d'erreur minimale.

On présente, en annexe, les preuves des propositions 5.1. et 5.2. Il en ressort que, si l'on cherche une RFS par simple localisation d'une mesure relativement à des bornes (simplifiant ainsi l'apprentissage et la génération du classifieur), sans détériorer les performances du système en négligeant une information disponible, alors il est nécessaire d'introduire dans la partition de l'espace de représentation une zone tampon entre domaine de sélection et domaine d'exclusion d'une classe donnée. Cette zone augmente la probabilité de non-classification, mais en même temps, elle introduit un écart en probabilité entre une classe sélectionnée et une autre exclue suffisamment grand pour que l'identification corresponde au maximum de vraisemblance.

On partitionnera le domaine K_i d'une caractéristique X_i relativement à chaque classe ω_j en trois sous-domaines :

- KS_{ij} : sous-domaine où ω_j est sélectionnée par X_i ;
- KE_{ij} : " " " " " exclue par X_i ;
- KN_{ij} : " " " " " non classée par X_i.

La partition finale de K_i est le produit des tri – partitions :

$$\bigtimes_{j=1}^{n} (KS_{ij}, KN_{ij}, KE_{ij}) = (k_{io}, K_{i1}, \ldots, k_{i,z_i-1}).$$

A chaque sous-domaine k_{il}, $0 \leq l \leq z_i - 1$, on associe une partition de Ω en trois sous-ensembles :

- $\Omega S_{il} = \{ \omega_j \mid k_{il} \subset KS_{ij} \}$
- $\Omega E_{il} = \{ \omega_j \mid k_{il} \subset KE_{ij} \}$
- $\Omega N_{il} = \{ \omega_j \mid k_{il} \subset KN_{ij} \}$

Il est possible de simplifier cette partition en regroupant en un seul sous-domaine tous les k_{il} tels que $\Omega S_{il} = \emptyset$. On peut également supprimer certains sous-domaines dont la probabilité d'y trouver une mesure est inférieure à un seuil : si

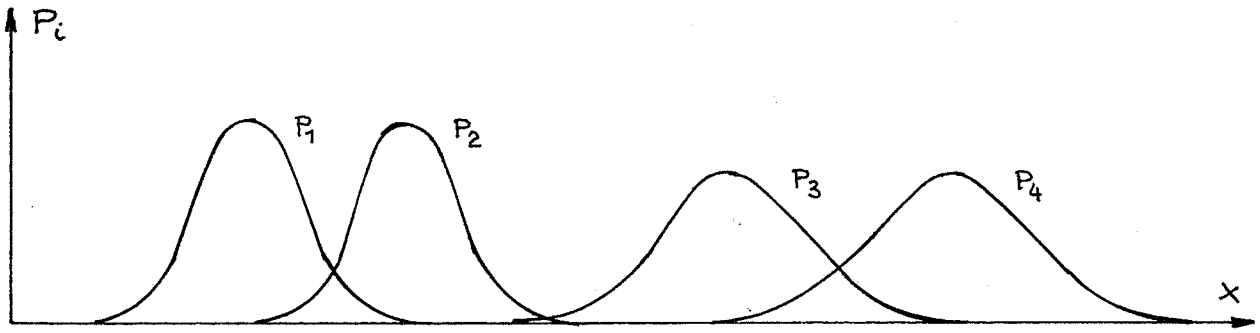
$$\sum_j P_r[\omega_j] \int_{k_{il}} P_{ij}(t)\, dt \leq \text{seuil.}$$

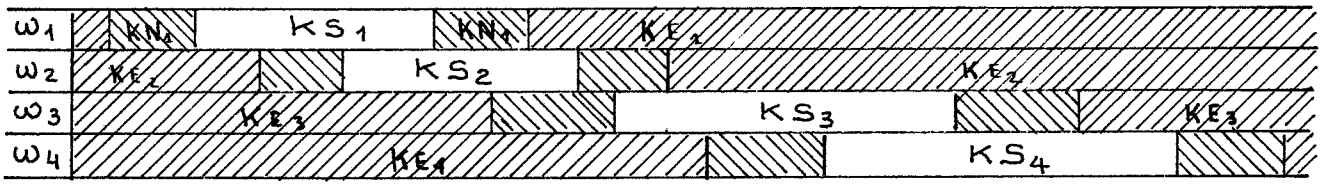
on regroupe k_{il} avec un sous-domaine voisin k_{iq} en $k'_{iq} = k_{iq} \cup k_{il}$ et

$$\Omega S'_{il} = \Omega S_{iq} \cap \Omega S_{il} \quad ; \quad \Omega E'_{il} = \Omega E_{iq} \cap \Omega E_{il}.$$

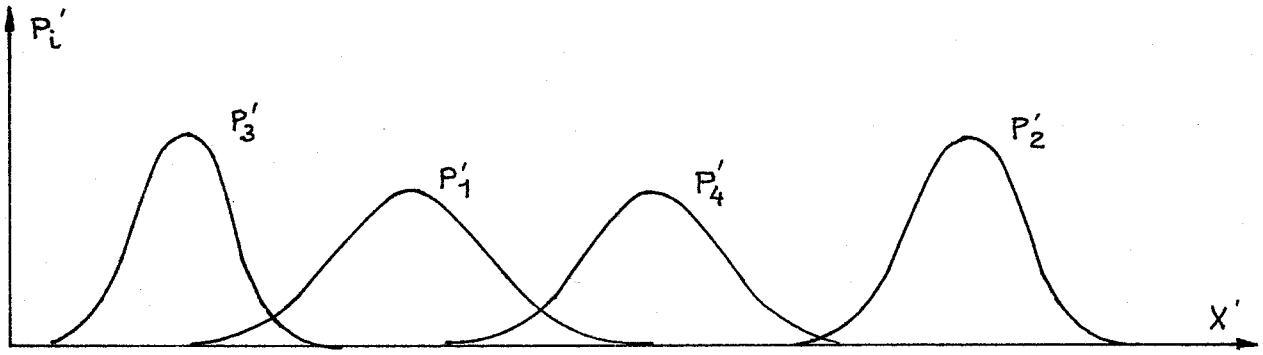
A partir de la partition résultante, on peut transformer le domaine d'application de X_i de l'ensemble K_i au segment d'entiers $Z_i = \{0, 1, \ldots, z_i - 1\}$; $X_i(e) = 1$ voulant dire $X_i(e) \subset k_{il}$. Cette transformation n'est pas un simple allègement des notations, elle correspond à l'abstraction d'une mesure entâchée d'incertitude en une information précise (mais plus pauvre), en vue d'une classification déterministe.

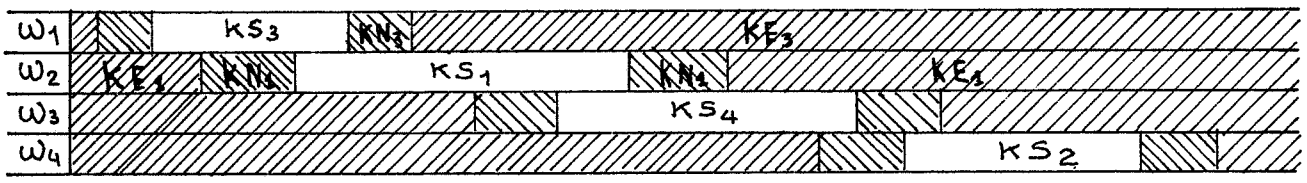
Exemple 5.2. : Illustrons cette méthode de segmentation sur un système de quatre classes d'objets $\{\omega_1, \ldots, \omega_4\}$ et deux caractéristiques X et X' de densités Gaussiennes. La figure 5.2. donne, pour chaque caractéristique, la tri-partition $(KS_{ij}, KN_{ij}, KE_{ij})$ relative à chaque classe et la partition produit avec les tri-partitions $(\Omega S_{il}, \Omega E_{il}, \Omega N_{il})$ correspondant à chaque sous-domaine (en regroupant en k_o et k'_o tous les sous-domaines pour lesquels $\Omega S = \emptyset$).





ΩS	ϕ	ϕ	ω_1	ω_1	ω_1,ω_2	ω_2	ω_2	ω_2	ϕ	ω_3	ω_3	ω_3	ω_3,ω_4	ω_4	ω_4	ϕ
ΩN	ϕ	ω_1	ϕ	ω_2	ϕ	ω_1	ω_1 ω_3	ω_3	ω_2 ω_3	ω_2	ϕ	ω_4	ϕ	ω_3	ϕ	ω_4
ΩE	Ω	ω_2 ω_3 ω_4	ω_2 ω_3 ω_4	ω_3 ω_4	ω_3 ω_4	ω_3 ω_4	ω_4	ω_1 ω_4	ω_1 ω_4	ω_1 ω_4	ω_1 ω_2 ω_4	ω_1 ω_2	ω_1 ω_2	ω_1 ω_2	ω_1 ω_2 ω_3	ω_1 ω_2 ω_3
		k_0	k_1	k_2	k_3	k_4	k_5	k_6	.	k_7	k_8	k_9	k_{10}	k_{11}	k_{12}	k_0





ΩS	ϕ	ϕ	ω_3	ω_3	ω_1 ω_3	ω_1	ω_1	ω_1	ω_1 ω_4	ω_4	ω_4	ω_4	ϕ	ω_2	ω_2	ϕ	ϕ
ΩN	ϕ	ω_3	ϕ	ω_1	ϕ	ω_3	ϕ	ω_4	ϕ	ω_1	ϕ	ω_2	ω_4 ω_2	ω_4	ϕ	ω_2	ϕ
ΩE	Ω	ω_1 ω_2 ω_4	ω_1 ω_2 ω_4	ω_2 ω_4	ω_2 ω_4	ω_2 ω_4	ω_3 ω_2 ω_4	ω_2 ω_3	ω_2 ω_3	ω_2 ω_3	ω_1 ω_2 ω_3	ω_1 ω_3	ω_1 ω_3	ω_1 ω_3	ω_1 ω_3 ω_4	ω_1 ω_3 ω_4	Ω
		k'_0	k'_1	k'_2	k'_3	k'_4	k'_5	k'_6	k'_7	k'_8	k'_9	k'_{10}	k'_0	k'_{11}	k'_{12}	k'_0	

FIGURE 5.2.

Finalement, un algorithme de RFS basé sur une telle partition serait :

Algorithme RFS

Données d'entrée : X, Ω, et e

1. Initialisation : $\Omega S \leftarrow \Omega$; $\Omega E \leftarrow \Omega N \leftarrow \emptyset$

2. Itérer tant que $[X \neq \emptyset$ et $\Omega S \neq \emptyset$ et $(|\Omega S| > 1$ ou $\Omega N \neq \emptyset)]$

2.1. Prendre et supprimer de X une caractéristique X_i

2.2. Evaluer $X_i(e)$, soit $1 \leftarrow X_i(e)$

2.3. $\Omega S \leftarrow \Omega S \cap \Omega S_{il}$

2.4. $\Omega E \leftarrow \Omega E \cup \Omega E_{il}$

2.5. $\Omega N \leftarrow \Omega - (\Omega S \cup \Omega E)$

2.6. Fin itération 2.

3. Fin

Données de sortie : Si $|\Omega S| = 1$ et $\Omega N = \emptyset$, alors classer e comme appartenant à l'unique classe de ΩS ;

Sinon : non-classification.

A tout moment $(\Omega S, \Omega N, \Omega E)$ est une 3 - partition de Ω. L'algorithme s'arrête dès que cette 3 - partition permet d'isoler une classe unique ω_j sélectionnée par toutes les caractéristiques évaluées, alors que toutes les autres classes ont été exclues au moins une fois (i.e., $\Omega S = \{\omega_j\}$ et $\Omega E = \Omega - \{\omega_j\}$). Il y a arrêt avec décision de non-classification si la condition précédente ne peut plus être satisfaite par la poursuite de l'algorithme : absence de caractéristique non encore évaluée, ou intersection des classes sélectionnées vide (i.e., $X = \emptyset$ ou $\Omega S = \emptyset$).

Pour (X_i, ω_j), on peut définir les domaines KS_{ij}, KE_{ij}, KN_{ij} par :

$\int_{KS_{ij}} p_{ij}(t)dt = \gamma$, et $\forall x,y : x \in KS_{ij}, y \notin KS_{ij} \Rightarrow p_{ij}(x) \geq p_{ij}(y)$

$\int_{KE_{ij}} p_{ij}(t)dt = \lambda$, et $\forall x,y : x \in KE_{ij}, y \notin KE_{ij} \Rightarrow p_{ij}(x) \leq p_{ij}(y)$

$KN_{ij} = K - KS_{ij} \cup KE_{ij}$, et $\gamma + \lambda \leq 1$

Il est évident que la segmentation par bi-partition correspond aux cas particuliers où $\gamma = 1 - \lambda$, i.e. $KN_{ij} = \emptyset$. Par ailleurs, on vérifie facilement que le majorant de la probabilité d'erreur (établie en Annexe I pour la segmentation par bi-partitions) :

$$Pr[\text{erreur}] \leq (m - 1)\lambda(1 - \lambda)^{r-1}$$

demeure valide dans le cas général. Il suffit de remarquer qu'en pire cas, on identifie e à ω_j sachant ω_i si ω_i a été sélectionnée (r-1) fois et exclue 1 fois.

Montrons, à présent, qu'il est possible de choisir γ et λ tels que toute identification de l'algorithme RFS corresponde au maximum de vraisemblance. Si les distributions sont Gaussiennes : on peut passer des paramètres γ et λ aux paramètres a et b (en notant $\varphi(x) = (1/\sqrt{2\pi})\, e^{-x^2/2}$; et $\Phi(x) = \int_0^x \varphi(t)\, dt$) :

$\Phi(a) = \gamma/2$ et $\Phi(b) = (1 - \lambda)/2$; i.e., le domaine de sélection sera : $KS_{ij} = [m_{ij} - a\sigma_{ij}, m_{ij} + a\sigma_{ij}]$, et le domaine d'exclusion $KE_{ij} =]-\infty, m_{ij} - b\sigma_{ij}] \cup [m_{ij} + b\sigma_{ij}, +\infty[$; m_{ij} et σ_{ij} étant les moyennes et écart type de P_{ij}.

Pour une classification dans ω_j, en pire cas :

- les r caractéristiques évaluées sélectionnent ω_j, mais leurs valeurs sont toutes aux limites des domaines de sélection $(m_{ij} \pm a\sigma_{ij})$;

- toute autre classe ω_l est au mieux exclue 1 seule fois, à la limite du domaine d'exclusion $(m_{ij} \pm b\sigma_{ij})$ et sélectionnée (r-1) fois, aux valeurs maximales des domaines de sélection (m_{ij}).

La classe ω_j correspondra au maximum de vraisemblance si :

$$\prod_{i=1}^{r} P_{ij}\,(m_{ij} \pm a\sigma_{ij}) \geq P_{ql}\,(m_{ql} \pm b\sigma_{ql}) \times \prod_{i \neq q} P_{il}\,(m_{il})$$

$$\varphi^{r}(a) \times \prod_{i=1}^{r} (1/\sigma_{ij}) \geq \varphi(b)\,\varphi^{r-1}(o) \prod_{i=1}^{r} (1/\sigma_{il})$$

Soit $M = \max_{jl} \left\{ \prod_{i=1}^{r} \sigma_{ij} / \sigma_{il} \right\}$; l'inégalité précédente sera vérifiée si : $(\varphi(a)/\varphi(o))^{r} / (\varphi(b)/\varphi(o)) \geq M$; i.e., $e^{(b^2 - ra^2)/2} \geq M$.

Pour tout r et M donné, un choix de a et b (et donc de α et λ) vérifiant cette inégalité est possible. Si par exemple l'on admet des écarts-types pour les différentes classes à produits à peu près équivalents, et M = 1, il suffit de prendre $b \geq a\sqrt{r}$. Ainsi, pour a = 1.5 et r = 7, une valeur acceptable serait b = 3.97 ; notons qu'alors les domaines de sélection, de non-classification et d'exclusion correspondraient respectivement à 86.7 %, 13.3 % et moins de 0.01 % des cas.

La généralisation au cas non Gaussien est simple. Soit :

$$P_s = \min \left\{ P_{ij}(x) \;\middle|\; 1 \leq i \leq n, \quad 1 \leq j \leq m, \quad x \in KS_{ij} \right\} , \text{ et}$$

$$P_e = \max \left\{ P_{ij}(x) \;\middle|\; 1 \leq i \leq n, \quad 1 \leq j \leq m, \quad x \in KE_{ij} \right\} .$$

Pour garantir une identification au maximum de vraisemblance, il suffit de prendre les domaines KS_{ij} et KE_{ij}, tels que :

$$P_s^{r-1} \geq P_e .$$

Les deux inégalités précédentes correspondent au cas le plus défavorable et imposent une même règle de segmentation pour toutes les caractéristiques d'une application donnée, indépendamment de leur pouvoir discriminant et des recouvrements plus ou moins importants entre leurs densités conditionnelles.

Pour certaines caractéristiques, il sera possible de réduire, ou même de supprimer les domaines de non-classification, tout en garantissant une identification au maximum de vraisemblance. On pourra procéder de la manière suivante :

- On fixe λ , et donc KE_{ij} en fonction de la contrainte sur la probabilité d'erreur : $\lambda = \eta/(m-1)$;

- On prend initialement γ à $\gamma = 1 - \lambda$, i.e., $KN_{ij} = \emptyset$ et $KS_{ij} = K - KE_{ij}$.

Pour chaque X_i, on détermine :

$$\alpha_i = \min \left\{ P_{ij}(x)/P_{il}(x) \;\middle|\; j,\ l \neq j,\ x \in KS_{ij} \cap KS_{il} \right\}$$

$$\beta_i = \max \left\{ P_{ij}(x)/P_{il}(x) \;\middle|\; j,\ l \neq j,\ x \in KE_{ij} \cap KS_{il} \right\}$$

Si $(\min_i \{\alpha_i\})^{r-1} \geq \max_i \{\beta_i\}$, l'identification au maximum de vraisemblance est garantie.

Sinon, on reprend la segmentation de la caractéristique X_i correspondant au $\min\{\alpha_i\}$ sur la base de $\lambda + \gamma < 1$, ce qui aura pour effet d'augmenter α_i et, accessoirement, de diminuer β_i. On peut poursuivre sur plusieurs caractéristiques (en augmentant à chaque fois la différence $1 - (\gamma + \lambda)$ précédente) jusqu'à ce que l'inégalité $(\min\{\alpha_i\})^{r-1} \geq \max\{\beta_i\}$ soit vérifiée.

Bien entendu, l'exigence que toute identification corresponde au maximum de vraisemblance se payera dans tous les cas d'une probabilité de non-classification d'autant plus importante que r est grand. En fait, il suffit de garantir que la probabilité d'une identification autre qu'au maximum de vraisemblance est négligeable par rapport à la contrainte η. Une procédure intégrant le calcul effectif de cette probabilité dans la définition des domaines de sélection et d'exclusion serait cependant d'une mise en oeuvre très lourde. On préconisera plutôt l'utilisation de la procédure précédente avec une règle d'arrêt heuristique sur l'écart $\left[\max\{\beta_i\} - (\min\{\alpha_i\})^{r-1}\right]$.

Terminons cette section en abordant brièvement la généralisation au cas d'une représentation non paramétrique ou pour des caractéristiques structurales.

Dans le cas non paramétrique, le nuage de points $X_i(e)$, $e \in \omega_j$, des échantillons d'apprentissage est représenté par un histogramme.

La méthode de segmentation de K_i est alors inhérente aux techniques statistiques utilisées. Dans le cas de l'application du L.A.A.S., une partition à priori en 10 à 20 segments d'égale longueur fut adoptée. On définira les domaines (KS_{ij}, KN_{ij}, KE_{ij}) d'une manière à peu près équivalente au cas paramétrique :

- on met dans KE_{ij} les segments de plus faible fréquence, et tels que la fréquence totale des points de KE_{ij} soit inférieure ou égale à λ . (En pratique, λ est très faible et on mettra dans KE_{ij} les segments où aucun point n'a été enregistré) ;

- on met dans KN_{ij} les segments adjacents à KE_{ij} (il y en aura deux pour un histogramme uni-modal) ;

- KS_{ij} comportera les segments restants.

On définit, par rapport aux fréquences f_{ij} des segments, les quantités

$$\alpha_i = \min \left\{ f_{ij}(s)/f_{il}(s) \mid l \neq j,\ s \text{ segment de sélection de } \omega_j \text{ et } \omega_l \right\}$$

$$\beta_i = \max \left\{ f_{ij}(s)/f_{il}(s) \mid l \neq j,\ s \in KN_{ij} \cap KS_{il} \right\} .$$

- Si $(\min\{\alpha_i\})^{r-1} \geq \max\{\beta_i\}$: arrêt ;
- Sinon, on modifie les partitions de certaines caractéristiques en ajoutant un segment au domaine KN_{ij}, entre KS_{ij} et KE_{ij}, et cela uniquement pour les classes ω_j dont l'histogramme est assez étalé, i.e., là où cette modification permettra d'augmenter α_i.

Notons que la définition de β_i utilise le majorant :

$$\Pr\left\{ e \in \omega_j \mid X_i(e) \in KE_{ij} \right\} \leq \max\left\{ f_{ij}(s) \quad s \in KN_{ij} \right\}.$$

A partir de là, on pourra définir les tri-partitions (ΩS_{il}, ΩN_{il}, ΩE_{il}) pour chaque segment k_{il} de K_i et effectuer les regroupements et simplifications sur la segmentation finale, comme précédemment.

Une caractéristique structurale (par exemple, relation d'inclusion entre contours, ou nombre de trous d'un objet) a pour domaine d'application un

ensemble discret à très faible cardinal (2 ou 3). Le problème de la segmentation ne se pose pas et la définition des sous-domaines relatifs aux diverses classes se fera par :

$KN_{ij} = \emptyset$

$KE_{ij} = \{$ valeurs de X_i pour lesquels aucune mesure sur les échantillons d'apprentissage n'a été enregistrée $\}$

$KS_{ij} = K - KE_{ij}$

5.2.3. GENERATION D'UN SYSTEME DE REGLES D'IDENTIFICATION

Les méthodes de la section précédente fournissent pour chaque caractéristique X_i :

- une transformation de son domaine d'application K_i en un domaine discret $Z_i = \{0, 1, \ldots, z_i - 1\}$, et

- une partition de Ω en $(\Omega S_{il}, \Omega N_{il}, \Omega E_{il})$ pour chaque valeur $l \in z_i$.

Ici, on cherche à définir les règles de décision du type : "Si $\left[(X_{i_1} = l_{i_1}) \text{ et } \ldots \text{ et } (X_{i_r} = l_{i_r})\right]$, alors $e \in \omega_j$" qui correspondent aux règles d'arrêt avec identification de l'algorithme SPR précédent : si SPR évalue $X_{i_1}, \ldots, X_{i_r}$ et trouve respectivement $l_{i_1}, \ldots, l_{i_r}$, alors il s'arrêtera en identifiant e à ω_j. En d'autres termes, cette règle sera valide si :

$$\bigcap_{i=i_1}^{i_r} \Omega S_{i l_i} = \{\omega_j\} \qquad \text{et} \qquad \bigcup_{i=i_1}^{i_r} \Omega E_{i l_i} = \Omega - \{\omega_j\}$$

On voudrait générer <u>toutes</u> les règles valides minimales pour la relation d'inclusion sur leur terme (<u>i.e.</u>, la règle précédente sera minimale si aucun sous-ensemble propre de conditions de $\{(X_{i_1} = l_{i_1}), \ldots, (X_{i_r} = l_{i_r})\}$ ne conduit à une règle valide).

Basiquement, on procèdera en ajoutant une condition $(X_i = l_i)$ à un terme candidat :

- si $\bigcap_i \Omega S_{i,l_i} = \emptyset$: on rejette le terme obtenu de toute considération ultérieure ;
- si $\left|\bigcap_i \Omega S_{i,l_i}\right| = 1$ et $\left|\bigcup_i \Omega E_{i,l_i}\right| = m-1$: on génère la règle d'identification correspondante ;
- sinon, on retient le terme obtenu comme candidat pour un développement ultérieur.

Dans le pire cas, la procédure examinera tous les k-tuples pour $1 \leq k \leq n$ et sera donc en $O(\prod_i (1 + z_i))$, mais, en fait, la règle d'élimination des K-tuples sera d'autant plus sélective et la procédure efficace, que les caractéristiques utilisées seront discriminantes, car alors peu de termes auront une intersection $\bigcap_i \Omega S_{i,l_i} \neq \emptyset$.

Exemple 5.3. : pour les quatre classes ω_1, ω_2, ω_3, ω_4 de l'exemple 5.2., les deux caractéristiques X et X' (figure 5.2) conduisent aux règles d'identification suivantes (présentées sous forme regroupée ; avec X et X' transformées sur le domaine $\{0,1,\ldots,12\}$:

"Si (X = 1) ou
(X'= 5) ou
[(X = 2 ou 3) et (X' = 3,4,6 ou 7)] alors e $\in \omega_1$"

"Si (X'=12) ou
[(X = 3,4,5 ou 6) et (X'= 11)] alors e $\in \omega_2$"

"Si (X = 8) ou
(X'= 1) ou
[(X = 7,9 ou 10) et (X'= 2 ou 3)] alors e $\in \omega_3$"

"Si (X = 12) ou
(X'= 9) ou
[(X = 10 ou 11) et (X'= 7,8 ou 10)] alors e $\in \omega_4$"

L'ensemble des règles $\{R_1, R_2, \ldots R_m\}$ obtenues (avec $R_j = \{u_{j1},\ldots,u_{jr}\}$: termes sur $(\bigtimes_i z_i)$ conduisant à l'identification de e dans ω_j) ne constitue pas un système $S = \{X, A, D, R\}$ complètement spécifié : il nous reste à définir D et à discuter de sa complétude. De plus, les règles possèdent des propriétés

particulières : les coordonnées indifférentes " φ " de leurs termes doivent être interprétées d'une manière restrictive et ces termes sont à recouvrement non vide.

Dépendance des caractéristiques : Le système de règles générées est apparemment inconsistant : certains termes de règles distinctes sont à intersection non vide. Ainsi, pour l'exemple précédent, des termes tels que (1,9) ou (12,5) sont des intersections de termes de R_1 et R_4.

Par définition des règles, si $u = v \cap w$, avec $v \in R_j, w \in R_l$ $l \neq j$, alors $\Omega S(u) = \emptyset$ et $\Omega E(u) = \Omega$; en notant :

$$\Omega S(u) = \bigcap_{i \notin \Phi(u)} \Omega S_{i,u(i)} \quad ; \text{ et } \quad \Omega E(u) = \bigcup_{i \notin \Phi(u)} \Omega E_{i,u(i)}$$

Puisque dans u toutes les classes sont exclues, la probabilité totale d'avoir un état e tel que $(X_1(e), \ldots, X_n(e)) \in u$ est inférieure à λ , et donc à la contrainte η sur la probabilité d'erreur.

Ceci nous autorise à admettre que tous les états correspondants à u sont quasi-impossibles, à η près, et que u est une relation de dépendance sur les caractéristiques plutôt qu'une inconsistance du système.

En définissant l'ensemble des dépendances par :

$$D = \{ u \mid \Omega E(u) = \Omega \}$$

l'ensemble de règles générées devient consistant.

Remarque : l'hypothèse d'indépendance conditionnelle des caractéristiques n'exclut pas, bien entendu, les dépendances dues aux données.

Complétude : Tous les termes de $(\bigtimes_i Z_i)$ pour lesquels l'algorithme RFS fournit une sortie de non classification ne sont pas couverts par l'ensemble de règles générées (pour l'exemple précédent des termes tels que :
(0,φ), (φ, 0), (2, 2), (2, 8), ou (2,10)).

On complètera le système par une règle Autre implicite correspondant à la non classification et dont tout terme u vérifie l'une des trois conditions :

. $\Omega S(u) = \emptyset$ et $\Omega E(u) \neq \Omega$

. $|\Omega S(u)| > 1$ et u est élémentaire (i.e. $\Phi(u) = \emptyset$)

. $|\Omega S(u)| = 1$ et $|\Omega E(u)| > m-1$ et u est élémentaire.

Interprétation des " φ " : La décomposition d'un terme $u \in R_j$ tel que $u(i) = \varphi$, en ses sous-termes $(u/i \leftarrow l)$ pour $0 \leqslant l \leqslant z_i - 1$, ne conduit pas nécessairement à z_i règles d'identification dans ω_j qui sont toutes valides. Autrement dit, si

$$\Omega S(u) = \omega_j \quad \text{et} \quad \Omega E(u) = \Omega - \{\omega_j\} ,$$

il n'est pas nécessaire que pour tout $k \in \Phi(u)$ et $0 \leqslant l \leqslant z_k - 1$, on ait :

$$\Omega S(u/k \leftarrow l) = \omega_j \text{ et } \Omega E(u/k \leftarrow l) = \Omega - \{\omega_j\} .$$

Ainsi, dans l'exemple précédent (5.3), pour la règle :

"Si (X = 1), alors $e \in \omega_1$"

seules les règles suivantes correspondent à une décomposition valide de $(1,\varphi)$:

"Si (X = 1) et (X' = 3,4,5,6, ou 7), alors $e \in \omega_1$",

alors que (1,0), (1,2) ou (1,8) correspondent à la règle Autre et que (1,1),(1,9), (1,10), (1,11) et (1,12) sont des relations de dépendances.

L'interprétation est la suivante : si le processus décisionnel évalue X(e) d'abord et trouve (X = 1), alors l'information disponible est suffisante pour identifier e à ω_1 (car la probabilité d'erreur dans l'état $(1,\varphi)$, qui est la somme pondérée des probabilités d'erreur dans chacun de ses sous-états, est inférieure à la contrainte) ; mais si l'on évalue d'abord X'(e), puis X(e) en trouvant (X' = 0,2 ou 8) et (X = 1), alors il n'est plus possible de négliger une partie de l'information obtenue et de classer e avec les garanties requises.

Notons que la probabilité de se trouver dans l'un des états correspondant à la règle Autre n'est pas négligeable (relativement à la contrainte), contrairement à celle de se trouver dans l'un des états de D.

Recouvrement des termes d'une même règle : (exemple : (1,5) ou (12,9))

Par définition, toutes les règles valides minimales ont été générées. S comporte donc explicitement tous les recouvrements possibles, compte-tenu de l'interprétation précédente des φ . On tiendra compte de ce fait avantageux pour la compilation de S en un arbre de décision.

5.2.4. GENERATION DU CLASSIFIEUR

La génération d'un arbre de décision ε - optimal pour l'application considérée pose quelques problèmes particuliers. Passons-les rapidement en revue.

Obtention des paramètres d'optimisation

La distribution de probabilité sur ($\bigtimes_i Z_i$) est donnée immédiatement par la connaissance :

- des probabilités à priori des classes (fréquences des objets à reconnaître) ;
- des densités conditionnelles des caractéristiques obtenues par apprentissage (avec ou sans l'hypothèse des densités paramétriques) :

$$\mu(u) = \sum_{j=1}^{n} P_r[\omega_j] \prod_{i=1}^{n} p(X_i = u(i) / \omega_j)$$

Le coût d'une caractéristique X_i correspond essentiellement au temps de calcul CPU du logiciel évaluant $X_i(e)$, en y incluant le temps de détermination de l'indice l du sous-domaine de K_i où se trouve la mesure $X_i(e)$ (lequel peut être ramené à la lecture d'une hash-table en O(1)).

Pour les caractéristiques de l'expérimentation du L.A.A.S., les mesures effectuées donnent des temps de calcul linéaires en nombre de points des contours et dépendants des caractéristiques précédemment évaluées (231) . Un modèle adéquat sera donc celui des coûts conditionnels et variables.

Eventuellement, la structure informatique de l'application peut imposer des coûts d'accès ; et, comme on le justifie plus loin, il sera avantageux d'introduire des coûts de maintenance.

Enfin, pour certaines caractéristiques (par exemple X_g = taux de remplissage, ou surface du contour externe divisée par la surface totale de ses trous), des contraintes de successions devront être introduites.

Génération de l'arbre

On utilisera les algorithmes développés au chapitre 3 et une fonction heuristique pour guider la recherche correspondante à un système avec règle Autre et pour le modèle de coût adopté. La seule difficulté réside dans l'interprétation restrictive des φ . Pour la résoudre, on restreindra, en conséquence, la définition d'un sous-système S/u par :

$$R_{j/u} = \left\{ u \cap v \;\middle|\; v \in R_j \text{ et } |\Omega S(u \cap v)| = 1 \text{ et } |\Omega E(u \cap v)| = m-1 \right\}$$

Exemple 5.4. : pour u =(3,φ) et S étant le système de l'exemple 5.3., on a :

$$R_{1/u} = \left\{ (3,3), (3,4), (3,5), (3,6), (3,7) \right\}$$

$$R_{2/u} = \left\{ (3,11), (3,12) \right\}$$

$$R_{3/u} = R_{4/u} = \emptyset$$

Compte-tenu du fait que tous les regroupements de termes valides sont présents dans le système de règles, le calcul des quantités q_j (probabilité d'arriver à une décision sans tester X_j) ne nécessitera pas la mise en oeuvre de la procédure RECOUVREMENT. Par contre, la transposition des formules de la proposition (3.3.2.) en : $q'_j = \sum_{u(j)=\varphi} \mu(u)$; somme sur tous les termes de S (compte-tenu des recouvrements des termes d'une même règle), ne correspond pas à q_j en général, mais en est un majorant : $q'_j \geq q_j$.

Cela est dû à l'interprétation restrictive des " φ " et au fait qu'un terme u tel que $u(j) = \varphi$ n'est équivalent qu'à un sous-ensemble de termes de $\left\{ (u/j \leftarrow l) \;\middle|\; o \leq l \leq z_j - 1 \right\}$.

Relativement à un sous-système S/v et à $q'_j(v) = \sum_{u(j)\neq\varphi} \mu(u, v)$, on a l'inégalité : $q'_j(v) \geq q_j(v)$.

Aussi, on reprendra pour heuristique h, la formulation développée dans le cas des contraintes de succession (Proposition 3.15) qui reposait également sur des majorants de q_j.

Problème des sorties de non-classification

Comme on l'a précisé en section (3.4.2.), dans le cas d'un système avec règle Autre, la procédure de compilation en arbre affecte à toute branche u, tel que $u \notin D$ et $S/u = \emptyset$, une feuille qui correspondra ici à une sortie de non-classification. Aussi, nous aurons de telles feuilles dès le premier rang de l'arbre et comme successeurs de pratiquement tous les noeuds.

Exemple 5.5. : Un arbre représentant le système de l'exemple 5.3. est donné en figure 5.3 (une branche non étiquetée correspond à toutes les valeurs de la caractéristique, autres que celles des branches étiquetées de même noeud).

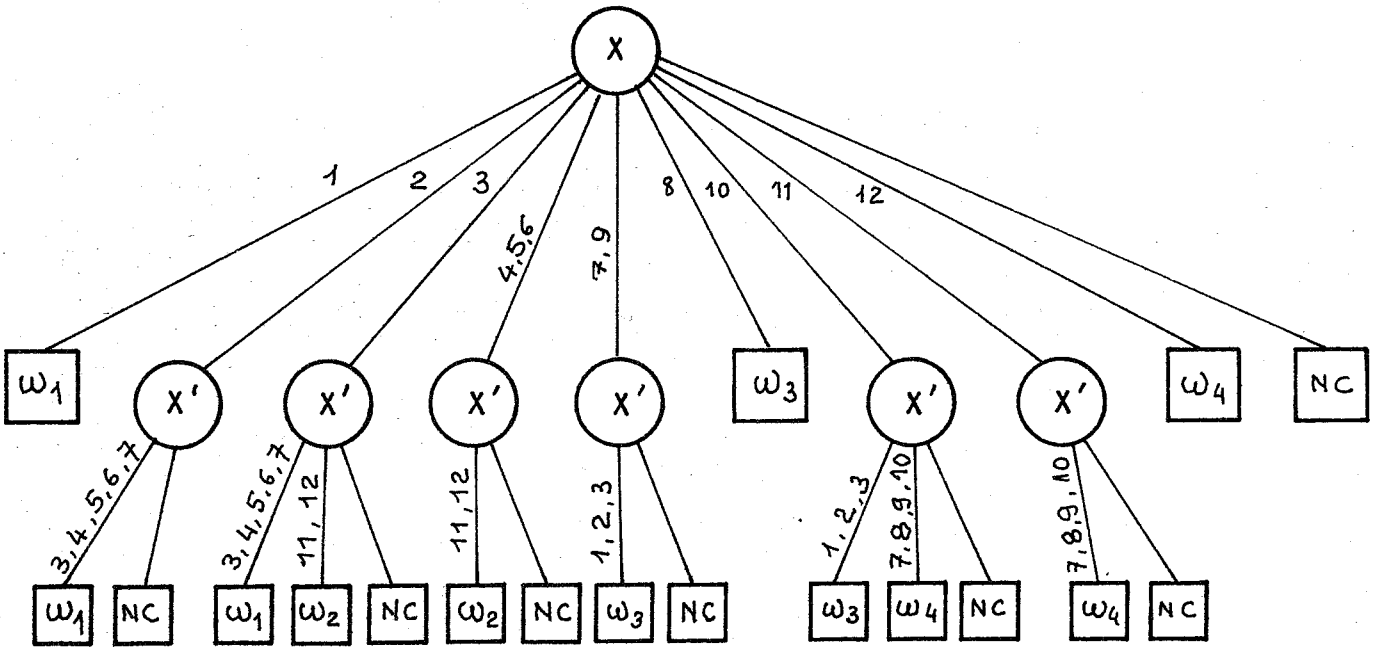


FIGURE 5.3.

Intuitivement, il n'est pas très satisfaisant de conclure à la non-classification d'un objet après une seule mesure (ou un nombre très faible), alors qu'il est possible d'acquérir davantage d'informations sur cet objet et, éventuellement, de l'identifier.

La règle de classification qui a été adoptée (une seule classe sélectionnée par toutes les caractéristiques, toute autre classe exclue par au moins une) simplifie considérablement la génération du classifieur et permet de garantir une bonne identification. Si la probabilité de non identification sur l'arbre de décision obtenu ($\sum \mu(u)$ sur les feuilles correspondantes) paraît élevée, on peut alors enrichir cette règle de classification moyennant un traitement supplémentaire sur les branches "Autre" de l'arbre.

Pour une branche associée à un terme u non élémentaire ($\Phi(u) \neq \emptyset$), on aura $\Omega S(u) = \emptyset$; aussi, on n'exigera plus qu'une seule classe soit sélectionnée par toutes les caractéristiques $\{ X_i \mid i \notin \Phi(u) \}$. On procèdera comme suit:

- Si $|\Omega E(u)| = m-1$, on calcule la probabilité d'erreur dans u pour une identification dans l'unique classe non encore exclue, et on conclut si cette probabilité est satisfaisante ;

- Sinon, on poursuit par l'évaluation d'autres caractéristiques (choisies heuristiquement) jusqu'à ce que toutes les classes soient exclues, sauf une pour laquelle la probabilité d'erreur est satisfaisante. Les termes ne remplissant pas cette condition (ou tout simplement, dont la probabilité est devenue très faible) seront associés à des feuilles de non-classification.

Dans le cas d'une branche Autre de rang n, u est élémentaire et il ne reste plus de caractéristique non encore évaluée. Néanmoins, on peut procéder comme ci-dessus lorsque $|\Omega E(u)| = m-1$, ou bien fournir toutes les classes sélectionnées si $|\Omega S(u)| \geqslant 1$ en vue d'une séparation par d'autres méthodes.

Remarques :

- Le sous-domaine de K_i dans lequel on avait regroupé tous les segments correspondants à $\Omega S_{i,1} = \emptyset$ (k_o et k'_o pour l'exemple 5.2)devra être séparé, pour ce traitement supplémentaire des feuilles de non-classification, suivant que $\Omega N_{i,1}$ est vide ou non ;

- Dans certains cas, on sera conduit à poursuivre le développement de chemins de l'arbre jusqu'au rang n. Pour éviter de retrouver les n caractéristiques du système présentes dans le classifieur, on pourra introduire des coûts de maintenance et un seuil sur la probabilité de passer par un chemin au-dessous de laquelle ce chemin est arrêté par une feuille de non-classification.

Terminons cette section en revenant sur les aspects de vérification après identification, et de dépendance conditionnelle des caractéristiques.

Vérification

Dans l'arbre généré, une feuille de non identification ne signifie pas que l'objet observé est inconnu et n'appartient à aucune des n classes de Ω. Tout au contraire, on suppose que e appartient nécessairement aux classes de Ω, et l'arbre se contente d'effectuer une discrimination (quand cela est possible) parmi ces classes.

Cette hypothèse n'est pas toujours justifiée. Ainsi, comme on ne teste pas que ce qui est observé est effectivement un objet unique, on pourra, par exemple, identifier deux objets de Ω en recouvrement comme étant un troisième objet.

En conséquence, après une identification de e à ω_j et localisation précise, une vérification sera nécessaire. Au mieux, elle se fera par une méthode de corrélation de motifs ("template matching") sur le contour externe, et/ou les trous de e et ω_j.

Dépendance conditionnelle

En Reconnaissance de Forme, l'indépendance conditionnelle est considérée en général comme une hypothèse forte et une simplification peu rigoureuse. Cependant, dans le cas du problème d'identification d'objets plans, on peut légitimement admettre qu'en dehors du bruit dû au système d'acquisition, la dispersion des objets d'une même classe est suffisamment faible pour justifier cette hypothèse pour presque toutes les caractéristiques. Une considération intuitive serait la suivante : si on connaît à priori quel est l'objet observé, la connaissance de son périmètre apporte peu d'information supplémentaire sur la valeur de sa surface.

On procèdera, bien entendu,au test quantitatif de l'hypothèse en estimant la matrice de corrélation sur la base des échantillons d'apprentissage et on comparera les coefficients de corrélation partiels (pour une classe donnée) à un seuil.

Si toutes les caractéristiques sont étroitement dépendantes, alors la méthode proposée ici n'est pas applicable ; et toutes les autres méthodes qui nécessitent l'estimation des densités de probabilités n - dimensionnelles ne le sont pas non plus, car cette estimation est en pratique difficilement réalisable.

Par contre, s'il y a très peu de dépendance, on pourra isoler certains groupes de caractéristiques et considérer chaque groupe comme une caractéristique unique. Si, par exemple X_i et X_j sont étroitement dépendantes, on supprimera chacune d'elle individuellement en les remplaçant par le produit X_iX_j. On estimera les densités bi-dimensionnelles et on procèdera à la partition du domaine $K_i \times K_j$ pour chaque classe de Ω en trois sous-domaines de sélection, d'exclusion et de non-classification de cette classe. On utilisera des droites orthogonales comme frontière de séparation (méthode de (88) par exemple). Comme précédemment, la partition finale de $K_i \times K_j$ sera le produit (simplifié) des 3-partitions. Le reste de la méthode s'appliquera sans changement.

Les limites de cette généralisation sont, bien entendu, dans la difficulté d'estimation des densités et la complexité des partitions des domaines k-dimensionnelles.

5.2.5. MISE EN OEUVRE EXPERIMENTALE

Plusieurs logiciels mettant en oeuvre diverses parties de l'application décrite ici ont été réalisés. Jusqu'à présent, ils n'ont été testés que séparément les uns des autres, et leur intégration en un système complet allant de la phase d'apprentissage à l'identification en ligne, n'a pas encore été achevée. Diverses difficultés ont retardé cette intégration, parmi lesquelles :

- plusieurs modifications, en cours de projet, du matériel informatique utilisé, alors qu'une partie des logiciels a été écrite en langage Assembleur;
- structure informatique complexe mettant en oeuvre plusieurs processeurs et inexistence des programmes système permettant de les interfacer.

Divers chercheurs du L.A.A.S. ont participé à la réalisation de ces logiciels. F. Stuck a développé (sur un système Mitra 15, couplé à un numériseur d'image Airazur) les programmes d'extraction des contours, d'évaluation des diverses caractéristiques présentées dans l'exemple 5.1., de localisation et de vérification dans la phase d'identification ; ainsi que des programmes conversationnels d'acquisition des échantillons d'apprentissage et de construction des histogrammes (Cf. (231)).

Ma contribution a porté sur la partie apprentissage, dans le cas d'une représentation paramétrique : à partir de la donnée des moyennes et écart-types des différentes densités, des paramètres de représentation et de différents seuils, un système comportant 75 fonctions APL (sur CIRCE) effectue la segmentation de l'espace de représentation, la définition du système de règles et la génération d'un arbre d'identification ε - optimal. Une partie de ce système a été reprise par les services techniques du L.A.A.S. en un programme PL1 traitant le cas non paramétrique (32).

La réalisation actuellement en cours d'intégration repose sur la structure suivante :

- identification en ligne : par extracteur câblé de contours en temps réel, couplé à un microprocesseur 16 bits qui évalue les caractéristiques en parcourant un arbre d'identification et effectue les tâches de localisation et de vérification (travail de J.C. TALOU) ;
- acquisition des données et apprentissage : par couplage du système précédent à un mini-calculateur 32 bits qui déterminera les diverses densités, définira le système de règles et génèrera (éventuellement en liaison avec un centre de calcul lourd) l'arbre qui sera fourni comme classifieur au processeur 16 bits.

Une expérimentation approfondie sera nécessaire pour valider les performances en lignes du classifieur, ainsi que la fiabilité de l'identification pour des ensembles d'échantillons d'apprentissage de taille limités (cf. (178)).

°0°

5.3. APPROCHE A UNE IDENTIFICATION SYNTAXIQUE D'OBJETS PARTIELLEMENT OBSERVES

Comme il a été mentionné en introduction, la méthode précédente reposant sur les caractéristiques globales d'un contour n'est utilisable que dans le cas d'objets isolés entièrement perçus. Dans de nombreuses applications, il est difficile ou même impossible de garantir cette dernière condition ; aussi la résolution du problème d'identification d'objets partiellement observés est d'un intérêt pratique important. De plus, cette résolution, dans le cas des hypothèses restrictives de l'identification d 'objets plans (dont on ne retiendra que H1 et H3), constituera un premier pas vers le problème plus général de l'identification tridimen$ionnelle.

L'identification d'objets partiellement observés relève de la Reconnaissance des Formes Syntaxiques. De très nombreuses publications lui ont été consacrées, parmi les plus significatives, on peut voir (8,16,51,52,173-175). Une synthèse critique de ces travaux dépasse le cadre de cette courte section, mais aucune méthode à ma connaissance ne résoud, d'une manière satisfaisante, les diverses phases (codage/représentation, classification, apprentissage) de ce problème encore ouvert.

On présentera ici, sous forme d'esquisse, les grandes lignes d'une approche en cours de développement, en mettant principalement l'accent sur les problèmes relevant de notre problématique ; et on développera une contribution algorithmique au problème de recherche de facteurs dans un mot.

Les aspects filtrage et représentation canonique d'un contour,relativement bien résolus,ne seront pas abordés (voir par exemple (31,156)).

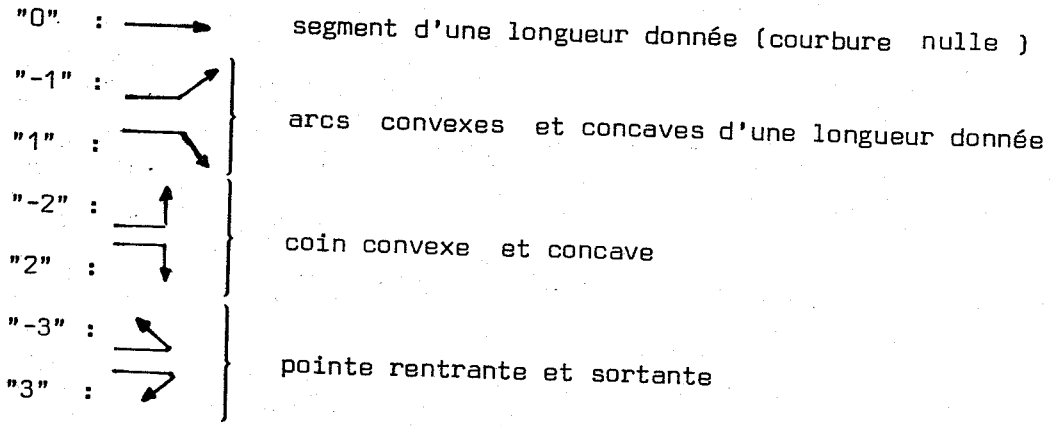
5.3.1. MODELE ET REPRESENTATION D'OBJETS PLANS

D'une manière générale, pour permettre une identification sur une vue partielle d'un objet, le modèle de représentation devra reposer sur des caractéristiques locales. Aussi bien ces caractéristiques que les relations qui les lient

dans ce modèle seront de deux types :

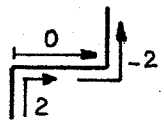
- type mono-dimensionnel le long d'un contour :
  - caractéristiques : parties spécifiques d'un contour externe, ou d'une ligne de niveau interne ;
  - relation : distance le long d'un contour entre deux caractéristiques mono-dimensionnelle ;
- type bi-dimensionnel :
  - caractéristiques : parties représentatives de la surface d'un objet (trou, détail de texture) ;
  - relation : position relative dans le plan entre deux caractéristiques mono ou bi-dimensionnelles.

En vue d'un apprentissage simplifié, on définira un alphabet permettant de représenter toute fonction de courbure discrétisée. On pourra prendre, par exemple, en s'inspirant du code de Freeman, l'alphabet $\Sigma = \{-3, -2, \ldots, +2, +3\}$, avec :

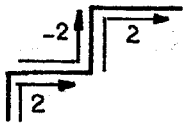


Il n'est pas nécessaire de prendre le même quantum d'abscisse curviligne pour toutes les lettres de cet alphabet. Typiquement, il sera de l'ordre de quelques (5 à 7) unités de distance élémentaire (inter-pixel).

Un contour donné peut être représenté par une concaténation de lettres de Σ en précisant les angles que forment les lettres entre elles. Certains auteurs utilisent un alphabet à attribut (235) , mais il est plus simple de décrire même ces angles par des lettres de Σ . Ainsi, les séquences "2 0 -2" et "2 -2 2" représentent respectivement :



et



Exemple 5.6. : La figure 5.3 donne le codage sur Σ d'une partie de contour en la séquence :

"20000020-11000-10-11-2000-1000-120012000-20-102".

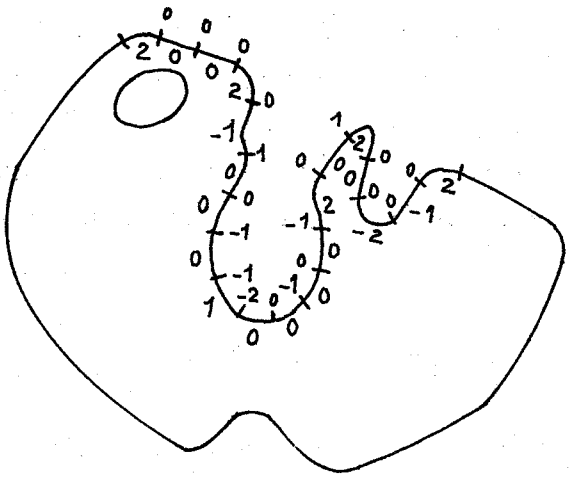


FIGURE 5.3.

Les avantages de cette représentation sont sa simplicité et son invariance à la rotation. Il reste à vérifier sa sensibilité au point initial de codage (diverses heuristiques, tel que choix d'un point d'inflexion, sont utilisables), à une légère translation ou homothétie et au bruit de quantification ; et à déterminer dans quelle mesure elle permet une unicité de représentation. Il reste également à préciser l'algorithme de codage, suivant qu'il porte sur un contour filtré ou non filtré.

A partir de là, on peut supposer que tout contour C_j est représentable par un mot (cyclique) M_j de Σ^*. Pour représenter un objet ω_j en termes de

caractéristiques locales mono-dimensionnelles, admettons que l'on dispose d'un ensemble de mots sur Σ , qu'on appellera primitives $P = \{ p_1, p_2, \ldots, p_r \}$. Le premier niveau du modèle de ω_j sera une décomposition de M_j en facteurs de P :

$$C_j \rightarrow M_j \rightarrow "p_{j1} \alpha_{j1} p_{j2} \cdots p_{jl} \alpha_{jl}"$$

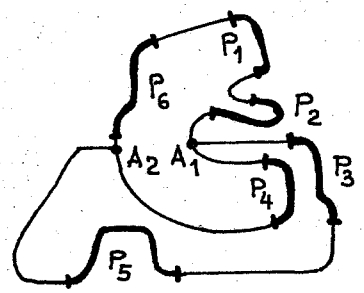
avec α_j : distance curviligne le long de C_j entre deux facteurs successifs de M_j (par exemple, entre les positions de leur première lettre).

Puisqu'on ne suppose plus que les objets sont reconnaissables uniquement à leur ombre, des lignes de niveau internes à un contour peuvent être également observées. L'hypothèse d'objets en recouvrement conduit à s'intéresser plus particulièrement aux lignes intersectant le contour externe. On suppose que les points communs à ces lignes et au contour sont accessibles. On codera également les lignes internes par des mots de Σ dont on cherchera une décomposition en facteur de P.

En adoptant certaines conventions de parcours et de codage des points d'intersection, le contour C_j et ses lignes internes pourront être représentés par un arbre étiqueté :

$$C_j \longrightarrow "p_{j1} \alpha_{j1} \; p_{j2} \; A_1 (\alpha'_{j2} \; p_{j3} \alpha'_{j3} \cdots A_2) \alpha_{j3} \cdots p_{jl} \alpha_{jl}"$$

Exemple 5.7. : Une représentation du contour de la figure 5.4 est :



$$"p_1 \alpha_1 p_2 \alpha_2 A_1 (\alpha_3 p_4 \alpha_4 A_2) \alpha_5 p_3 \alpha_6 p_5 \alpha_7 A_2 \alpha_8 p_6 \alpha_q"$$

A_1 et A_2 étant les points d'intersection entre la ligne interne et le contour.

FIGURE 5.4.

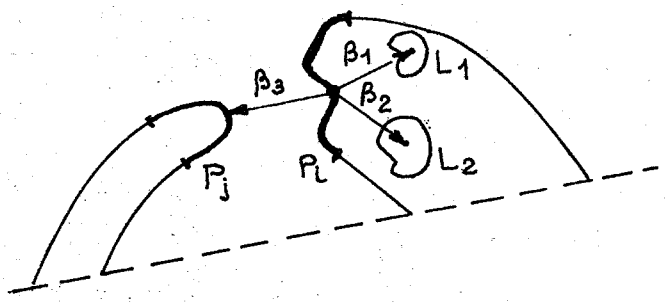
Remarque : L'aspect cyclique ne pose pas de problème pour le contour externe, parcouru toujours dans le même sens. Par contre, une ligne interne pourra être parcourue dans l'un des deux sens. Ainsi, une représentation équivalente pour l'exemple précédent sera :

"$P_5 \alpha_7 A_2(\alpha_4 P_4 \alpha_3 A_1)\alpha_8 P_6 \alpha_9 P_1 \alpha_1 P_2 \alpha_2 A_1 \alpha_5 P_3 \alpha_6$"

Notons, cependant, que si l'on tient compte de la symétrie miroir, alors même les contours externes pourront être lus dans l'un des deux sens.

Le dernier niveau du modèle de représentation qu'on préconise dans cette approche porte sur les caractéristiques et relations bidimensionnelles. Soit $\{L_1, L_2, \ldots L_q\}$ un ensemble de motifs bidimensionnels donné. On mettra en relation une primitive d'un niveau quelconque de la représentation précédente avec les motifs et les primitives dans son voisinage (surfacique), en précisant leurs positions et orientations relatives.

Exemple 5.8. :



Une partie de la représentation du contour de la figure 5.5. sera :

"$\ldots\alpha_i p_i[(\beta_1 L_1) ; (\beta_2 L_2) ; (\beta_3 P_j)]\ldots$"

FIGURE 5.5.

Chaque p_j ou L_i étant une entité orientée, la relation du type "β" sera donnée,par exemple, par deux angles et une longueur.

Notons, finalement, qu'aussi bien pour les primitives mono-dimensionnelles que pour les motifs bidimensionnels, la présence d'une caractéristique dans un contour sera définie à un seuil près sur une distance à préciser. Dans le cas des primitives, on peut prendre une distance de Hamming du type $\sum_i |a_i - b_i|$, $a_i, b_i \in \Sigma$. Pour tout ce qui suit, on dira que p_i est facteur de M_j, ou que L_i est présente dans C_j, en sous entendant que la distance correspondante est inférieure au seuil.

5.3.2. APPRENTISSAGE

L'apprentissage porte, bien entendu, sur des objets isolés entièrement perçus, et donc éventuellement sans tenir compte des lignes de niveau internes.

On admet que l'ensemble des échantillons d'une classe ω_j peut être moyenné en un contour unique C_j, lequel sera codé par un mot M_j de Σ^*.

Informellement, le problème de l'apprentissage peut se poser de la manière suivante :

1) Déterminer un ensemble de primitives $P = \{ p_1, \ldots, p_r \}$, de telle sorte que la décomposition de chaque M_j en facteur de P, "$p_{j1} \alpha_{j1} \ldots p_{jl}$", constitue un ensemble de caractérisations locales de M_j. On veut dire par là, qu'au mieux chaque p_{ji} n'est facteur d'aucun mot autre que M_j. Plus vraisemblablement, plusieurs sous-séquences courtes (2 à 3 facteurs) de "$p_{j1} \alpha_{j1} \ldots p_{jl} \alpha_{jl}$" ne figureront dans la décomposition d'aucun autre mot que M_j ;

2) Chercher dans le "voisinage" de chaque p_{ji} de la décomposition "$p_{j1} \alpha_{j1} \ldots p_{jl} \alpha_{jl}$" de M_j s'il existe à l'intérieur du contour des caractéristiques bidimensionnelles de type donné à priori, ou bien sur le contour lui-même des primitives p_{jk} (y compris celles $p_{j,i-1}$ ou $p_{j,i+1}$ si les longueurs $\alpha_{j,i-1}$ ou α_{ji} dépassent un seuil), et définir les positions relatives de ces caractéristiques par rapport à p_{ji}. On en déduira la représentation de M_j :

"$p_{j1} [(\beta_{j1} L_{j1}) ; (\beta'_{j1} L'_{j1}) ; \ldots] \alpha_{j1} p_{j2} \ldots p_{jl} \alpha_{jl}$" ;

3) Générer, à partir de ces représentations, un ensemble de règles du type :

"Si une partie d'un contour comporte la séquence

"$p_{i1} [(\beta_1 L_1), (\beta_2 L_2), \ldots] \alpha_{i1} p_{i2} [\ldots] \ldots$", alors $e \in \omega_j$".

Une approche à la première tâche pourrait être la suivante :

- Si F est un facteur de M_1 de largeur l donnée (de l'ordre de 5 lettres), on cherchera toutes les positions où F est facteur du mot $M_2 M_3 \ldots M_m$

  . si le nombre de ces positions dépasse un seuil, on abandonne cette recherche et le facteur F correspondant ;

  . sinon, on essaie de réduire ce nombre en augmentant F à gauche et/ou à droite par les lettres qui lui sont adjacentes dans M_1.

On aura ainsi une liste des facteurs spécifiques à M_1, des facteurs qui figurent dans un seul autre mot que M_1, de ceux qui figurent dans deux autres mots, ... etc, en se restreignant par exemple à m/2.

- La même opération sera répétée pour M_2, M_3, ..., M_m, et l'ensemble des facteurs obtenus constituera P.

Au total, la procédure s'effectuera, au pire des cas, en $O(N^2)$ si N est le nombre total de caractères de $M_1 M_2 \ldots M_m$; moyennant certaines heuristiques de sélection à priori des facteurs qui seront testés (notamment sur la fréquence de lettres), elle pourrait être ramenée à une complexité plus faible.

La principale difficulté de la phase 2 porte sur la définition à priori (et indépendamment d'un ensemble de classes d'objets) des types de motifs bidimensionnels qui seront utilisés. On devra également définir un seuil sur le voisinage de chaque primitive dans lequel sera restreinte la recherche, et sur le nombre maximal de relations bidimensionnelles qu'on autorisera.

On peut aborder la phase 3 en se restreignant tout d'abord un ensemble de séquences discriminantes sans relations bidimensionnelles. Partant des représentations "$p_{j1}\alpha_{j1} \ldots p_{jl}\alpha_{jl}$", pour $1 \leq j \leq m$, des m classes, et connaissant les mots autres que M_j où figure chaque p_{ji}, on procèdera d'une manière similaire à la phase 1. On se limitera à des séquences ne couvrant pas plus d'une fraction (par exemple 1/3) d'un contour, et on introduira une autre distance pour mesurer

l'écart entre une séquence "$p_{ji}\alpha_{ji}$ ..." et une partie d'un contour $M_k \neq M_j$. La recherche de ces séquences discriminantes se fera également en temps polynômial en M.

Les relations bidimensionnelles peuvent n'être ajoutées qu'à titre de vérification sur les séquences obtenues. Mais si l'on est plus ambitieux sur le type de règle, admettant une discrimination sur la base des relations bidimensionnelles, et de plusieurs séquences, chacune non discriminante (tel que, par exemple, la règle : "si le contour comporte la séquence "$p_{i1}[(\beta_1 L_1) ou (\beta_2 L_2)]\alpha_{i2}p_{i2}$" et la séquence "$p_{j1}\alpha_{j1}p_{j2}$ ..." ..."), alors la synthèse de l'ensemble de règles sera beaucoup plus difficile et complexe. Il en va de même si l'on veut prendre en compte les lignes de niveau internes propres à un objet donné. Les représentations des objets seront alors des arbres étiquetés et l'apprentissage consistera à en chercher un ensemble de sous-arbres discriminants.

5.3.3. IDENTIFICATION

Le processus d'identification des objets correspondants à un contour donné comportera (dans un ordre non séquentiel) les quatre opérations suivantes :

1) Recherche des primitives de P figurant dans un contour et dans ces lignes internes ;

2) Recherche des caractéristiques bidimensionnelles internes à ce contour ;

3) Séparation des objets perçus sur la base d'une interprétation des lignes internes ;

4) Recherche dans chaque partie de contour entre deux points d'intersection, et dans les lignes internes de séquences discriminantes correspondantes à des règles d'identification.

L'opération (1) peut se faire avec ou sans codage préalable des contours en lettre de Σ . Bien que ce choix doive être fait sur la base des performances de l'ensemble du processus d'identification (et,en particulier l'opération (4)), il semble à priori plus avantageux de procéder comme suit :

- lors de la phase d'apprentissage,effectuer le codage inverse de chaque primitive p_i de P en une liste de points de la fonction de courbure correspondante ;

- lors de la phase d'identification, déterminer les occurrences de p_i dans un contour, par corrélation, entre les points de la fonction de courbure associée au contour et ceux correspondants à p_i.

Cette détermination peut se faire :

- soit par une recherche de chaque primitive individuellement, qui sera combinée à la recherche de séquences discriminantes. L'ordre des primitives sera donné par une arborescence binaire obtenue par compilation des règles d'identification ;

- soit par une recherche simultanée de toutes les occurrences de toutes les primitives dans un contour, qui s'effectuera en O(lqr), avec : l = nombre de points dans le contour, q = nombre de points d'une primitive, r = |P|. La recherche des séquences discriminantes se fera alors sur la base des informations obtenues.

On résoudra le problème de la séparation des objets perçus par une méthode d'"hypothèse - confirmation". La principale difficulté réside dans la distinction entre une ligne interne correspondant au recouvrement de deux objets, et celle propre à un objet donné.

Finalement, on cherchera les caractéristiques bidimensionnelles conformes aux relations "β" des séquences discriminantes obtenues et vérifiant la séparation des objets retenue.

°0°

5.3.4. ALGORITHMES DE RECHERCHE DE FACTEURS DANS UN MOT

La recherche de facteurs dans un mot est un problème commun à de nombreuses applications en informatique. Les algorithmes qui vont suivre ne sont pas spécifiques à celle développée dans ce chapitre ; aussi, sont-ils présentés dans une formulation générale.

Soit Σ^* le monoïde libre engendré par l'alphabet Σ de $\sigma = |\Sigma|$ lettres; et T et M deux mots de Σ^* de longueur respective $t = |T|$ et $m = |M|$. On note
T(i) : $i^{ème}$ lettre de T
T(i, i+1, ... i+j) = T(i) T(i+1) ... T(i+j) séquence des (j + 1) lettres de T à partir de la $i^{ème}$ position.

On définit l'ensemble (ordonné) des occurrences de M dans T par :

$$Occ(M,T) = \left\{ 1 \leqslant i \leqslant t-m+1 \mid T(i, i+1, \ldots i+m-1) = M \right\} ;$$

et on dit que M est un facteur de T si Occ (M,T) $\neq \emptyset$.

En général, T est un "texte"avec $t \gg m$,et on cherche à savoir si le mot M apparaît dans T. Plus précisément, on peut se poser les problèmes suivants :

- P1 : M est-il un facteur de T ?
- P2 : Quel est l'ensemble Occ(M,T) ?
- P3 : On se donne T et plusieurs mots $M_1, \ldots, M_s$; et on cherche à résoudre P1 ou P2 pour l'ensemble de ces mots par rapport à T.
- P4 : On autorise une certaine dissemblance entre un facteur et la partie du texte qui lui correspond : on introduit une lettre $\varphi \notin \Sigma$; M et/ou T appartiennent à $(\Sigma \cup \{\varphi\})^*$, et on définit :

$$Occ(M,t) = \left\{ 1 \leqslant i \leqslant t-m+1 \mid \forall j \; ; \; 1 \leqslant j \leqslant m : T(i-1+j) = M(j) \text{ ou } M(j) = \varphi \text{ ou } T(i-1+j) = \varphi \right\}.$$

Un algorithme traitant P1 peut être à la base d'une résolution, non seulement de P2 - P4, mais aussi d'autres problèmes dérivés, parmi lesquels on peut citer :

- le facteur bidimensionnel : T et M sont des matrices sur Σ , et on généralise la définition de l'occurrence de M dans T (7) ;

- le facteur d'un arbre : T et M sont des arbres étiquetés sur Σ , chaque lettre ayant une arité fixée (nombre de branches issues du noeud correspondant) et on cherche les positions où M est un sous-arbre de T (92) ;

- le plus grand facteur commun à deux mots (4,91) .

On s'intéressera donc initialement au problème P1 et on traitera quelques généralisations par la suite.

La procédure "naïve" cherche la concordance avec M à partir du premier caractère de T, abandonne dès qu'une erreur est trouvée et reprend à partir du caractère suivant de T. Elle est très inefficace et nécessite, en pire cas, un nombre de comparaisons quadratiques en O(mt).

L'algorithme qu'on propose ici est une généralisation des deux principaux algorithmes linéaires connus dans la littérature : celui de Knuth, Morris et Pratt (124) et celui de Boyer et Moore (18) . Le premier compare M à T dans l'ordre gauche-droite ; le deuxième dans l'ordre droite-gauche ; celui que nous proposons génère pour chaque M un ordre optimal de comparaison.

L'algorithme de (124) est basé sur l'idée suivante : dans le cas d'une comparaison de M à T séquentielle (dans l'ordre croissant des indices), si on trouve dans une certaine position :

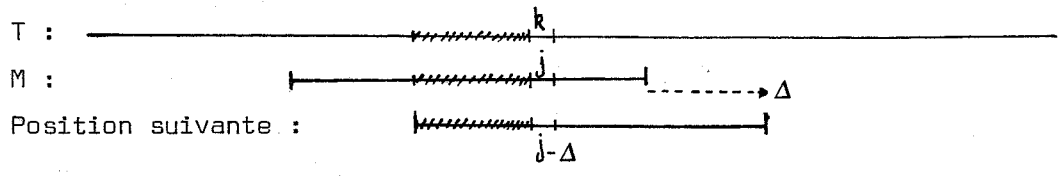
$M(j) \neq T(k)$ (i)

alors, on a nécessairement :

$M(1,2, \ldots, j-1) = T(k-j+1, \ldots, k-1)$ (ii)

En reprenant la comparaison dans une position où M est décalée par rapport à T de Δ caractères, il ne pourra y avoir occurrence que si :

$M(1,2, \ldots, j - \Delta) = T(k - (j - \Delta), \ldots, k)$ (iii)



Par transitivité (sur la partie hachurée), (i) - (iii) impliquent :

$$\left.\begin{array}{l} M(\Delta + 1, \ldots, j - 1) = M(1, \ldots, j - \Delta - 1) \\ M(j - \Delta) \neq M(j) \end{array}\right\} \quad \text{(iv)}$$

Si l'on connaît la valeur <u>minimale</u> de Δ vérifiant (iv), alors à la suite d'un test négatif $M(j) \neq T(k)$, il suffit de décaler M de Δ positions par rapport à T, et de reprendre la comparaison à partir du test $M(j - \Delta) \stackrel{?}{=} T(k)$, <u>sans revenir en arrière dans T</u>.

Or, on voit dans (iv) que cette valeur minimale ne dépend que de M et de j. On pourra, par un prétraitement de M, générer une table $\Delta(1), \ldots, \Delta(j), \ldots, \Delta(m)$ grâce à laquelle l'algorithme de recherche de la première occurrence de M dans T sera le suivant :

<u>Algorithme KMP</u>

<u>Données d'entrée</u> : T, M, table Δ

1. <u>Initialisation</u> : $j \leftarrow 1$; $k \leftarrow 1$ (j pointeur sur M, k pointeur sur T)

2. <u>Itérer</u> tant que [$j \leq m$ <u>et</u> $k \leq t$]

2.1. <u>Si</u> $T(k) = M(j)$, <u>alors faire</u> : $j \leftarrow j+1$; $k \leftarrow k+1$

2.2. <u>Sinon Si</u> $\Delta(j) = j$, <u>alors faire</u> : $j \leftarrow 1$; $k \leftarrow k+1$

<u>Sinon</u> $j \leftarrow \Delta(j) - j$

3. <u>Fin</u>

<u>Données de sortie</u> : <u>Si</u> (j = m) ; sortir (k-m+1) : position de la première occurrence de M dans T

<u>Sinon</u> (k = t et $j \neq m$) : M n'est pas un facteur de T.

On estime traditionnellement la complexité de ce type d'algorithmes en nombre total de comparaisons de deux caractères, en supposant que M n'est pas un facteur de T. Ici, chaque itération incrémente k, ou bien décrémente j (i.e., avance M par rapport à T), et ceci peut être fait au plus t fois par indice : l'algorithme KMP effectue au plus (2t) comparaisons. Mais, il est important de voir qu'il effectuera au moins t comparaisons, chaque caractère de t étant adressé au moins une fois.

Knuth et al (124) présentent plusieurs analyses et extensions de leur algorithme, parmi lesquelles :

- la compilation des données M et Δ directement en langage machine, avec deux instructions par comparaison ;

- la résolution des problèmes P2 et P3 ;

- et surtout l'utilisation du même algorithme pour générer la table Δ en temps linéaire en O(m) (analyse de la réoccurrence de M(1,2, ..., j - Δ - 1) dans M(Δ + 1, ..., j - 1)).

L'algorithme de Boyer et Moore (18) est basé sur le même principe : prétraitement de M pour exploiter les réoccurrences de sous-séquences de M dans lui-même. Mais, alors que KMP compare M à T séquentiellement de gauche à droite, l'idée originale de cet algorithme est de partir à chaque fois du caractère le plus à droite de M, et de procéder vers la gauche (en décrémentant les pointeurs j et k).

Un avantage très substantiel en résulte : un test négatif dans une certaine position peut permettre d'économiser les comparaisons sur tous les caractères à gauche de cette position. Ainsi, pour T(k) $\neq$ M(m), on pourra déplacer M par rapport à T vers la première position (à partir de la droite) où T(k) apparaît dans M. En particulier, si T(k) est un caractère qui ne figure pas dans M, on reprendra directement à partir de T(k+m).

Exemple 5.9: pour $\Sigma = \{a, b, c, d\}$ et les deux mots ci-dessus, la séquence des positions successives de M par rapport à T, si on les compare dans l'ordre droite gauche, est la suivante (on souligne les caractères testés) :

T : b c b a b a b d a b c d a b c a b c b b a ...

M : b c b c b b

b c b c b b

b c b c b b

b c b c b b

Un algorithme (légèrement amélioré par rapport à celui de (18)) utilisera une table bidimensionnelle donnant le déplacement de M par rapport à T, si un test négatif $T(k) \neq M(j)$ se produit. Cette table sera définie par :

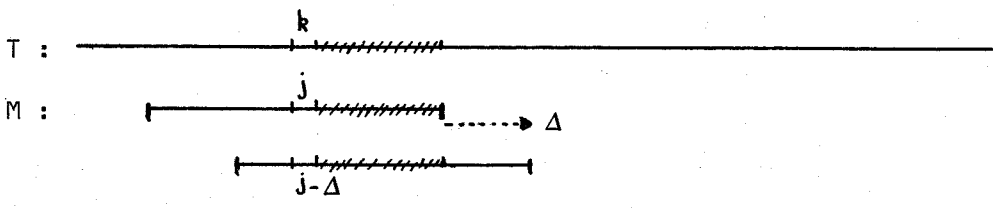
$$(\alpha, j) \in \Sigma \cup \{1, \ldots, m\} \longrightarrow \Delta(\alpha, j) \text{ avec}$$

- $(\alpha, j) = o$ si $\alpha = M(j)$; sinon
- $(\alpha, j) = \min \left\{ 1 \leq \delta \leq m \mid (M(j-\delta) = \alpha \text{ ou } \delta \geq j) \text{ et } (\forall i,\ j < i \leq m :\ M(i-\delta) = M(i) \text{ ou } \delta \geq i) \right\}$

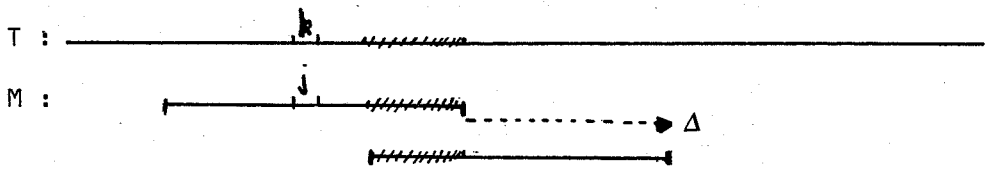
Comme pour l'algorithme KMP, la table Δ est définie de façon à ce que les caractères de M qui concordent avec ceux de T, réoccurrent dans M avec un décalage Δ. Ainsi, les conditions :

$(\forall i \quad j < i \leq m : \quad M(i - \delta) = M(i) \quad \text{ou} \quad \delta \geq i)$

correspondent respectivement aux deux cas de figure (a) et (b) :



(a)



(b)

Cependant, on exige, de plus ici, que le caractère $\alpha = T(k)$ corresponde à $M(j - \delta)$ (cas a), ou bien que le décalage $\Delta(\alpha,j)$ dépasse $T(k)$ (cas b).

Exemple 5.10 : Pour le mot M de l'exemple précédent, la table Δ est :

	b	c	b	c	b	b
Σ \ j	1	2	3	4	5	6
a	5	5	5	5	5	6
b	0	5	0	5	0	0
c	5	0	5	0	1	2
d	5	5	5	5	5	6

Remarquons que a et d ne figurant pas dans M, on a pour tout j :

$\Delta(a,j) = \Delta(d,j)$.

Connaissant Δ , la recherche de la première occurrence de M dans T procèdera simplement suivant l'algorithme :

Algorithme BM

Données d'entrée : T et Δ

1. Initialisation : $j \leftarrow m$; $k \leftarrow m$

2. Itérer tant que $(j > 0$ et $k \leq t)$

2.1. Si $\Delta(T(k),j) = 0$, alors faire : $j \leftarrow j-1$; $k \leftarrow k-1$

2.2. Sinon faire :

$k \leftarrow k + \Delta(T(k),j) + m-j$

$j \leftarrow m$

3. Fin

Données de sortie : Si $(j = 0)$, sortir $(k+1)$

Sinon : M n'est pas un facteur de T

Cet algorithme effectue deux types d'itérations :

- examen de M et de T dans l'ordre droite-gauche, lorsqu'une concordance est constatée (on décrémente j et k en 2.1) ;

- décalage relatif de M par rapport à T de Δ positions en cas de test négatif (k est incrémenté de $\Delta + (m - j)$). et reprise de l'examen à partir du caractère le plus à droite ; $j \leftarrow m$).

Cette dernière opération correspond à un saut dans T : l'algorithme BM contrairement à KMP, n'adresse pas tous les caractères de T. Par contre, après avoir effectué un saut, BM oublie les caractères de T précédemment adressés, et peut les tester de nouveau lors des parcours droite-gauche suivants, et cela plusieurs fois. Il s'ensuit que la preuve de la linéarité de BM est loin d'être aussi simple que celle de KMP.

La plus intéressante est due à Guibas et Odlyzko (79) , qui développent, à propos de cet algorithme, plusieurs théorèmes sur les périodes des mots (P est période de M si $M = P'P^k$ avec $P = P'P''$; et on vérifie facilement que $\forall \alpha, j$, si $\Delta(\alpha,j) \neq 0$, alors les $\Delta(\alpha,j)$ derniers caractères de M sont une période de $M(j - \Delta + 1, \ldots, m)$). La présentation de l'algorithme qui est faite ici (une table Δ bidimensionnelle unique, au lieu de deux tables monodimensionnelles) permet de simplifier légèrement leur preuve. Le résultat est que le nombre Y de comparaisons effectuées par BM est majoré par 4t.

On a montré que Y peut atteindre 2t (sur l'exemple suivant : $M = b\ a\ b\ a\ a(b\ a)^k$; et $T = b\ b(a\ a\ b\ a\ a(b\ a)^{k-1}b)^r a$, on vérifie que $Y = 2t \times (k+6/4-1/4r)/(k+1/2+6/4r)$, Y tend asymptotiquement vers 2t avec k). Il est vraisemblable que (2t) soit le majorant effectif de Y.

Notons finalement que la génération de Δ s'effectue en $O(\sigma' m)$; $\sigma' = \min\left\{\sigma,\ 1+ \text{ le nombre de caractères distincts de M}\right\}$.

L'intérêt pratique de BM n'est pas sa linéarité en pire cas, mais plutôt son comportement moyen "sous-linéaire". En effet, Boyer et Moore proposent

une étude théorique et expérimentale de leur algorithme, d'où il ressort (en admettant que tous les mots M sont équiprobables et tous les caractères de T des variables aléatoires indépendantes équidistribuées) que le nombre moyen de comparaisons est inférieur à t et décroît avec T et m. Par exemple, pour l'alphabet latin et m = 14, moins de (t/10) comparaisons sont effectuées. Selon leur analyse, ce bon comportement est dû essentiellement au fait que dans le test $T(k) \stackrel{?}{=} M(m)$, le plus souvent le caractère T(k) n'appartiendra pas à M (pour T grand), ce qui permettra à l'algorithme de faire, pour une seule comparaison, un saut maximal de $m = \Delta(T(k),m)$ caractères.

L'algorithme que nous proposons part de l'idée suivante : si la distribution des lettres dans T n'est pas uniforme, alors la position j dans 1, ..., m pour laquelle le test $T(k) \stackrel{?}{=} M(j)$ a le plus de chance d'être négatif ne correspond pas nécessairement à j = m. La position suivante la plus avantageuse n'est pas non plus j = m-1 ; et l'ordre droite-gauche, même s'il améliore le comportement moyen de BM par rapport à KMP, n'est pas le meilleur ordre de comparaison possible pour <u>chaque</u> M. Ceci est de plus vrai même lorsque la distribution des lettres est uniforme.

On formulera donc la phase de prétraitement de M comme un problème d'optimisation. Il s'agit de trouver, pour un mot M donné, un ordre des lettres de M de façon à ce que la recherche de la première occurrence de M dans un texte T quelconque effectue, en moyenne, un nombre minimal de comparaisons, si elle compare les lettres de M à ceux de T suivant cet ordre. Précisons cette procédure de recherche.

Soit r une permutation de 1,2, ..., m , r(i) étant l'entier en $i^{ème}$ position dans r. Par commodité, on convient de noter : r(m+1) = 1.

Définissons une table bidimensionnelle Δ_r, pour un mot M et une permutation r données par :

$$\Delta_r(\alpha,j) = 0 \quad \text{si } M(r(j)) = \alpha \quad ; \text{ sinon}$$

$$\Delta_r(\alpha,j) = \min \left\{ 1 \leq \delta \leq m \;\middle|\; (M(r(j)) - \delta) = \alpha \ \underline{\text{ou}}\ \delta \geq r(j)) \ \underline{\text{et}} \ (\forall i,\ 1 \leq i < j : M(r(i) - \delta) = M(r(i)) \text{ ou } \delta \geq r(i)) \right\}$$

Algorithme RO (Recherche Ordonnée)

Données d'entrée : Δ_r, T, r

I. Initialisation $j \leftarrow 1$; $k \leftarrow r(1)$

2. Itérer tant que $(j \leq m$ et $k \leq t - m + r(1))$

2.1. Si $\Delta_r(T(k),j) = 0$, alors faire :

$k \leftarrow k + r(j+1) - r(j)$

$j \leftarrow j + 1$

2.2. Sinon faire :

$k \leftarrow k + \Delta_r(T(k),j) + r(1) - r(j)$

$j \leftarrow 1$

3. Fin

Données de sortie : Si $(j = m+1)$, alors sortir (k) position de la première occurrence de M dans T ;

Sinon : M n'est pas un facteur de T.

Exemple 5.11. : Illustrons le comportement de l'algorithme RO pour M = b d d d b b b b c a et r = (8,9,10,7,6,5,4,3,2,1).

La table $\Delta_r(\alpha,j)$ est donnée ci-dessous :

M	b	d	d	d	b	b	b	b	c	a
r(j)	10	9	8	7	6	5	4	1	2	3
j	1	2	3	4	5	6	7	8	9	10
a	10	10	10	10	10	10	10	8	9	0
b	0	10	10	10	0	0	0	0	1	9
c	10	10	10	10	10	10	10	8	0	10
d	10	0	0	0	10	10	10	4	7	10

L'ordre des comparaisons avec les caractères de T et la séquence des déplacements de M est :

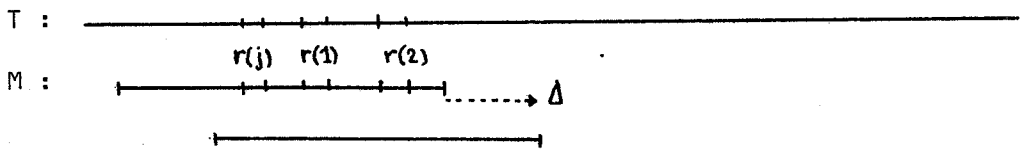
```
T : a b c b c d a b c a b c a b c d a b b c a.....
                  4 1 2 3
M : b d d d b b b b c a
                              5 6
                    b d d d b b b b c a
                                11 10 7 8 9
                      b d d d b b b b c a
```

Proposition 5.4. : Pour toute permutation r donnée, l'algorithme RO est correct.

Preuve : Par récurrence sur la séquence des déplacements de M.

. Base : Par définition $\Delta(\alpha,j) = 0$ lorsque $M(r(j)) = \alpha$; donc pour la position initiale de M, on effectuera la suite de tests : $T(r(1)) \stackrel{?}{=} M(r(1))$, $T(r(2)) \stackrel{?}{=} M(r(2))$, ... etc, et cela jusqu'à ce que :

- ou bien $j = m + 1$, auquel cas les m tests précédents ont tous été positifs, la première occurrence de M est en position 1, et la donnée de sortie est : $k = r(1) + (r(2) - r(1)) + \ldots + (1 - r(m)) = 1$;
- ou bien $\Delta(T(r(j)),j) \neq 0$, pour un j : $1 \leq j \leq m$, on a alors le schéma suivant :



Le déplacement de M par rapport à T de Δ caractères assure :

1) La concordance avec le caractère : $T(r(j)) = M(r(j) - \Delta)$, ou que la position suivante ne recouvre pas $T(r(j))$;

2) la concordance avec tous les caractères $M(r(1)), \ldots, M(r(j-1))$ pour lesquels on a eu précédemment un test positif :
$\forall i, 1 \leq i < j$: $M(r(i) - \Delta) = M(r(i))$ ou $\Delta \geq r(i)$; et

3) aucun autre déplacement inférieur à Δ ne garantit (1) et (2).

Donc aucune occurrence ne peut se trouver à gauche de la deuxième position de M dans T.

Notons que dans ce déplacement, la valeur du pointeur k est passée de r(j) à (Δ + r(1)) ; ce qui fait reprendre les tests en position suivante : $T(\Delta + r(1)) \stackrel{?}{=} M(r(1))$, ... etc.

. Induction : L'absence d'occurrence à gauche de la $i^{ème}$ position de M dans T permet de reprendre le même raisonnement que précédemment. De plus, la séquence des déplacements de M s'arrête sitôt que k + (m- r(1)) t, i.e., dès que le déplacement suivant amènera le dernier caractère de M à l'extérieur de T ; et dans ce cas, M n'est pas un facteur de T. □

Il est remarquable de voir à quel point les algorithmes RO et BM sont similaires. La recherche dans un ordre quelconque n'est pas plus compliquée que dans l'ordre séquentiel droite-gauche : à l'exception de l'adressage d'une table en (RO : 2.1.1.), une itération de RO entraîne le même nombre d'opérations qu'une itération de BM (on intègre, bien entendu, dans le prétraitement les opérations indépendantes de T : (α,j) + r(1) - r(j) et r(j+1)-r(j)). Le comportement linéaire, en pire cas de BM, n'est cependant pas conservé par RO pour tout ordre r (sur $M = a^n b$, $T = a^n b$, et r = (1.2, ..., n+1), on retrouve la procédure naïve en $O((n+1)^2)$). Mais, la complexité de RO ne doit être jugée que pour l'ordre r optimal relatif à un mot M. Abordons cette optimisation.

Soit $\{p(\alpha) \mid \alpha \in \Sigma\}$, la distribution de probabilité des lettres de dans un texte $T \in \Sigma^*$. On admet l'hypothèse d'indépendance des lettres de T. On notera, pour simplifier :

$p_j = p(M(r(j)))$; et $p_o = 1$.

Pour un mot M donné, on cherche un ordre r tel que l'algorithme RO effectue, en moyenne (sur l'ensemble des textes dont les lettres vérifient la distribution précédente), un nombre minimal de comparaison par lettre de T, lorsque M n'est pas facteur de T. D'une manière équivalente, on cherchera à maximiser

le critère inverse : saut moyen maximal par comparaison :

$$\Psi(r) = \frac{\sum_{1 \le j \le m} p_1 p_2 \cdots p_{j-1} \sum_{\alpha} \Delta_r(\alpha, j)\, p(\alpha)}{\sum_{1 \le j \le m} j\, p_1 p_2 \cdots p_{j-1} (1 - p_j)}$$

Le problème général de la recherche de la permutation r qui maximise Ψ est NP-Complet (Sahni (214) : "Permutation functions problem"). De plus, la formulation du problème en une recherche arborescente (choix de r(1) en la racine, puis choix de r(2) en le noeud suivant,...etc)ne bénéficiera pas d'un critère additif, sauf lorsque les lettres de Σ sont équidistribuées, auquel cas, le dénominateur de $\Psi(r)$ ne dépend pas de r :

$$\sum_{j=1}^{m} j\, p^{j-1}(1-p) = (1 - p^m(m+1-mp))/(1-p)$$

On remarque, dans ce cas particulier, que le terme associé au choix de r(j) décroît très rapidement avec j :

à r(1) correspond : $(1-p) \sum_{\alpha} \Delta_{r(1)}(\alpha, 1)$

à r(j) correspond : $p^{j-1}(1-p) \sum_{\alpha} \Delta_{r(1)\ldots r(j)}(\alpha, j)$

Comme dans tous les cas $\Delta_r(\alpha, j) \le m$, le terme associé à r(j+1) sera du $j^{\text{ème}}$ ordre (en base $\sigma = 1/p$) par rapport à celui de r(1), et donc rapidement négligeable.

On exploitera cette remarque pour déterminer, linéairement en m et σ' un ordre r sous-optimal constituant une bonne approximation. On procèdera par une recherche bornée dans l'arborescence précédente, guidée par l'estimation heuristique f(r(1),..., r(k)) de $\Psi(r)$ pour tout r commençant par r(1), ..., r(k) :

$$f(r(1), \ldots, r(k)) = \frac{\left[\sum_{1 \le j \le k} p_1 \cdots p_{j-1} \sum_{\alpha} \Delta_r(\alpha, j)\, p(\alpha)\right] + m\left[\sum_{k+1 \le j \le m} p_1 \cdots p_k\, \bar{p}^{\,j-k-1}(1-\underline{p})\right]}{\left[\sum_{1 \le j \le k} j\, p_1 \cdots p_{j-1}(1-p_j)\right] + \sum_{k+1 \le j \le m} j\, p_1 \cdots p_k\, \underline{p}^{\,j-k-1}(1-\bar{p})}$$

avec $\bar{p} = \max_j \{p_j\}$ et $\underline{p} = \min_j \{p_j\}$

L'estimateur f est majorant et coïncidant.

Preuve très simple : $\forall r = (r(1),\ldots, r(k), \ldots) : \Delta_r(\alpha, j) \leq m$, et

$\Sigma_{k+1\leq j\leq m} p_1 \cdots p_k \cdots p_{j-1} \Sigma_\alpha \Delta_r(\alpha,j) p(\alpha) \leq m \Sigma_{k+1\leq j\leq m} p_1 \cdots p_k \bar{p}^{j-k-1}(1-\underline{p})$; et

$\Sigma_{k+1\leq j\leq m} j p_1 \cdots p_k \cdots p_{j-1}(1-p_j) \geq \Sigma_{k+1\leq j\leq m} j p_1 \cdots p_k \underline{p}^{j-k-1}(1-\bar{p})$ □

Il est donc possible de mettre en oeuvre rigoureusement un algorithme A_ϵ, avec le test suivant pour borner l'exploration : on arrête la recherche en un noeud $(r(1),\ldots, r(k))$ si l'écart relatif de $f(r(1),\ldots,r(k))$ à $\Psi(r)$, pour r obtenu en complétant les (m-k) valeurs restantes dans le sens décroissant, est inférieur à .

En pratique, on a implémenté une simple descente en profondeur limitée au rang 3 . Comme l'optimalité n'est pas garantie, on compare le $\Psi(r)$ obtenu à celui de l'ordre droite-gauche, et on adopte le meilleur. La génération de r et de Δ_r est alors, à une constante multiplicative près, du même ordre de complexité que la génération de Δ pour l'algorithme BM.

Le comportement moyen de l'algorithme RO sera toujours supérieur ou égal à celui de BM. Le gain par rapport à cet algorithme dépend, bien entendu, du masque M et de la distribution de probabilité sur Σ . A titre d'illustration pour le mot M de l'exemple précédent et une distribution uniforme des quatre lettres a, b, c, d , l'ordre r = (8,9,10 7,6, ..., 1) permet à RO d'effectuer (t/5) comparaisons en moyenne, alors que BM en nécessitera (t/3.56), soit 40,4 % de plus. Bien entendu, la distribution uniforme n'est pas favorable à RO : sur le même exemple, une distribution telle que (.54, .01, . 44, .01) conduit la recherche en profondeur à prendre l'ordre r = (8,7,4,10,9,6,5, ..., 1), grâce auquel RO ne fera que (t/7.87) comparaisons, alors que BM continuera à en faire (t(3.32), soit 2.37 fois plus.

Nous avons implémenté ces algorithmes et testé leur comportement en pratique, sur des textes et des mots générés aléatoirement.

Les conclusions d'une expérimentation réduite (4 tailles d'alphabet σ = 4,7,12 et 25 ; 4 à 5 tailles de mots de 7 à 20 lettres, textes de 5000 caractères, 20 essais par points) sont les suivantes :

- les comportements moyens mesurés et prédits concordent très étroitement ;
- le gain de RO par rapport à BM augmente rapidement avec la taille des mots, puis se stabilise ; ainsi, par exemple pour $\sigma = 4$, on trouve respectivement des gains moyens de 3, 10, 17 et 22 % pour m = 7,10,15 et 20 ;
- ce gain diminue avec la taille de l'alphabet : pour m = 15 et σ=4,7,12 et 25 lettres, on trouve respectivement un avantage moyen de 17,11,8 et **3** %.

Notons que les distributions de probabilités utilisées (elles-mêmes générées aléatoirement) étaient très faiblement dispersées (par exemple, en comparaison à la distribution des lettres de l'alphabet latin pour un texte en anglais, où le rapport des probabilités extrêmes dépasse 150, mais où, par contre l'hypothèse d'indépendance est erronée). Il n'est pas exclu que, pour des distributions plus contrastées le gain moyen ne soit plus important.

Comment se situe notre algorithme RO par rapport à un algorithme optimal ? Par un argument très simple Knuth et al (124) établissent l'existence d'un algorithme optimal dont la complexité moyenne est en O(t log m/m). Plus récemment, Yao (243) montre, par preuve plus élaborée, que (t log m/m) est aussi un minorant de la complexité moyenne de tout algorithme (alors qu'un minorant de la complexité, en pire cas, est (t-m+1) selon Rivest (203)).
La preuve de Yao suppose qu'on ne teste jamais deux fois le même caractère de T. Or, l'algorithme RO, tout autant que BM, oublie, à chaque décalage de M les tests précédemment effectués (voir exemple 5.11 : tests 6-7 et 5-10). Ceci explique que malgré l'optimisation de r, $\Psi(r)$ restera inférieur au maximum théorique qu'est (m/log m).

Nous avons donc élaboré un algorithme qui reprend le même principe que RO, mais, qui de plus, se souvient des positions précédemment testées. On précalcule quelles seraient, après un décalage Δ, les positions dans la séquence $\Delta + r(1)$, $\Delta + (r(2)$ ... etc, qui correspondent à celles déjà testées en r(1), r(2), ... etc. De même, on détermine les recouvrements possibles après deux décalages , 3, ... etc, jusqu'à ce que $\sum_j \Delta \geq m$. A l'issue de ce prétraitement chaque élément de la table Δ n'est plus un entier unique, mais une liste de sauts successifs.

Le prétraitement est assez complexe. L'algorithme lui-même doit gérer plusieurs pointeurs et listes de décalages, et calculer les sauts successifs (vers la doite ou vers la gauche) par comparaison de ces listes. Cela réduit son intérêt pratique, bien que minimal relativement au nombre moyen de caractères comparés (critère à prendre avec précaution).

Dans l'implémentation que nous avons effectuée, la comparaison à l'algorithme BM ne s'est révélée favorable à l'algorithme "optimal" que pour un alphabet binaire (i.e., là où BM est inefficace, car les déplacements sont faibles et les recouvrements importants). Mais, dans ce cas, il y a intérêt, plutôt que d'effectuer les comparaisons individuellement, à exploiter les facilités de comparaison en parallèle sur plusieurs bits existantes dans presque tout processeur. On ne détaillera pas plus avant cet algorithme.

Passons en revue, à présent, quelques généralisations du problème P1.

La recherche de toutes les occurrences de M dans T (problème P2) est immédiate en recommençant à partir de chaque occurrence. Galil (53) présente une amélioration de BM dans ce cas, qui précalcule un $(m+1)^{ème}$ saut à effectuer après une occurrence. On gagne quelques comparaisons si M apparaît fréquemment dans T. Cette amélioration est directement applicable à RO.

La recherche de plusieurs facteurs M_1, M_2, ..., M_s dans T (problème P3) est plus intéressante. Karp, ainsi que Aho et Corasick (3) proposent une généralisation de KMP qui procède ainsi : on lit T séquentiellement de gauche à droite, chaque caractère une et une seule fois, un caractère lu entraîne un changement d'état d'un automate fini (ayant autant d'états que de préfixes distincts $M_i(1, \ldots, j)$, $1 \leq j \leq \max\{mi\}$). L'algorithme est en O(t) et le prétraitement en $O(\sigma \sum_i mi)$.

Nous avons proposé une généralisation des algorithmes BM et RO reprenant l'idée de sauts dans T pour réduire le nombre moyen de comparaisons. Présentons-là, informellement sur un exemple, en prenant pour simplifier l'ordre droite-gauche.

Exemple 5.12 : Soient les mots sur $\Delta = \{a, b, c, d\}$, que nous alignons sur le caractère le plus à droite :

M_1 = a b b c

M_2 = b c a c

M_3 = a b a b c

M_4 = a b c a b

Ces mots sont prétraités en l'arbre de comparaison et de saut suivant :

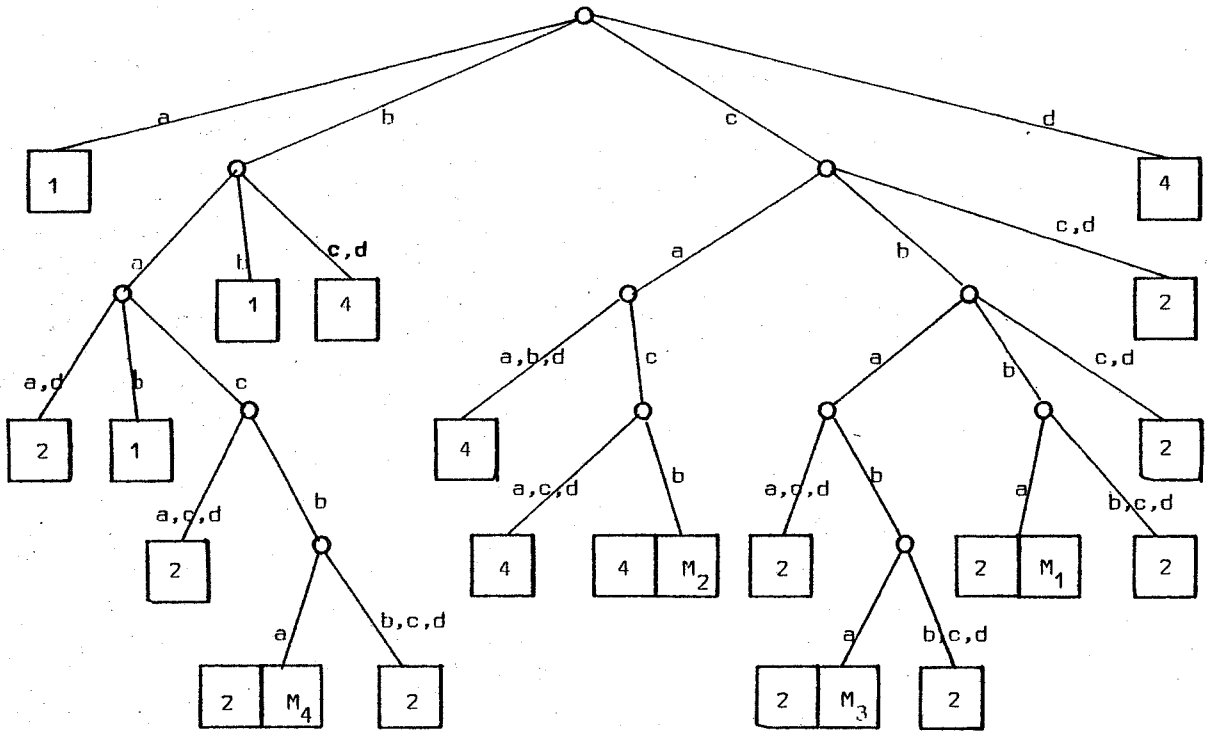


FIGURE 5.6.

On procède, à partir du caractère $\min_i \{ mi \}$ de T, et de la racine de l'arbre. Chaque caractère lu fait prendre la branche correspondante. Si cela mène à un noeud interne, on lit le caractère suivant (dans l'ordre droite-gauche) de T. Si elle mène à une feuille étiquetée par l'entier Δ, on déplace tous les masques par rapport à T de Δ positions (le pointeur sur T est incrémenté de Δ - 1 + rang de la feuille) ; éventuellement, on note l'occurrence du mot M_i présent dans la feuille. L'ordre des tests sur T est :

T =	b	b	c	a	c	c	b	a	c	b	a	a	b	d	b	a	b	a	b	c	a	b	...
		5	4	1 3	2		8	7	6		11	10	9	13	12	21	16 20	15 19	14 18	17			
		M_2														M_3							

La génération d'un tel arbre se fera d'une manière similaire à celle d'une table de saut. Soit M le mot associé à un chemin de l'arbre d'une branche de rang k à la racine. S'il existe un mot M_i tel que $M = M_i\ (m_i - k+1, \ldots, m_i)$, alors :

- si $m_i > k$: la branche sera transformée en un noeud interne ;
- si $m_i = k$: elle sera transformée en une feuille associée à l'occurrence de M_i ;

Sinon, elle correspondra à une feuille étiquetée uniquement par un entier Δ, calculé comme étant le déplacement minimum compatible avec l'un quelconque des masques M_i (réoccurrence de M dans M_i décalé de Δ).

Il est simple, une fois l'arbre généré, de déterminer le saut moyen par comparaison qui lui correspond. Dans le cas de l'exemple précédent, on vérifie que pour la distribution de probabilité sur Σ : (0.4, 0.1, 0.1, 0.4), l'algorithme présenté effectuera (t/2.088) comparaisons en moyenne, soit 2 fois moins que celui séquentiel de gauche à droite.

On peut généraliser l'approche de RO à la détermination d'un ordre de comparaison optimal pour chaque chemin de l'arbre. A une branche seront alors associés une étiquette correspondant au caractère lu, et un entier relatif indiquant le déplacement du pointeur sur T pour le caractère suivant à lire. Le

calcul du saut en une feuille respectera l'ordre du chemin qui y mène. On cherchera l'arbre maximisant le rapport du saut moyen à la hauteur moyenne.

Cette recherche est évidemment plus coûteuse que celle d'un ordre unique nécessaire à RO. Aussi, on ne la recommandera que dans le cas d'un aplhabet faible, à distribution contrastée et pour un ensemble de masques utilisés très intensivement. Autrement, on préconisera la génération de l'arbre correspondant à l'ordre droite-gauche (cet arbre a au plus m_i noeuds internes, un noeud par suffixe distinct $M_i(j, j+1, \ldots, m_i)$ pour $j \geqslant 2$, $1 \leqslant i \leqslant s$), le calcul du saut moyen correspondant, et l'adoption de l'approche Karp-Aho-Corasick si ce saut est inférieur à 1.

Abordons finalement le cas où M et/ou T contiennent le caractère particulier " φ ", considéré comme étant équivalent à toute lettre de Σ (problème P4). (cf. (44)).

Aussi bien pour l'algorithme KMP que pour les algorithmes BM et RO, la définition du déplacement Δ repose sur la transitivité de la relation d'égalité entre deux caractères. Or, la relation d'équivalence considérée ici n'est pas transitive. Aussi, l'algorithme KMP, qui ne revient pas en arrière dans T pour vérifier l' équivalence après décalage, n'est plus utilisable. Par contre,l'inconvénient de BM et de RO d'oublier les tests précédemment effectués devient ici un avantage qui permettra leur généralisation sans difficulté.

On définira l'équivalence dans $(\Sigma \cup \{\varphi\})$ par :

$\alpha \equiv \beta$ ssi $(\alpha = \beta)$ ou $(\alpha = \varphi)$ ou $(\beta = \varphi)$.

On reprend les définitions de la table Δ en remplaçant chaque relation d'égalité par une équivalence. Pour l'ordre droite-gauche, on aura par exemple :

$\Delta(\alpha, j) = 0$ si $M(j) \equiv \alpha$ (et donc, en particulier $\Delta(\varphi, j) = 0, \forall j$); sinon

$$\Delta(\alpha, j) = \min\left\{ 1 \leqslant \delta \leqslant m \;\middle|\; (M(j - \delta) \equiv \alpha \text{ ou } \delta \geqslant j) \text{ et } (\forall i,\ j < i \leqslant m : M(i - \delta) \equiv M(i) \text{ ou } \delta \geqslant i) \right\}.$$

On vérifie qu'avec cette définition, BM et RO restent valides sans autre modification.

Remarque : Le travail présenté ici a été effectué en 1978 - 1979, exposé à divers séminaires, mais n'a jamais été publié. On m'a transmis récemment une étude due à Commentz-Walter (25) qui reprend l'idée d'une exploration droite-gauche dans le cas de plusieurs masques ; son algorithme utilise un arbre (trie) similaire à celui présenté ici, il en montre la non-linéarité en pire cas et propose une modification qui le rend linéaire, mais moins performant en moyenne. Jusqu'à présent, je n'ai eu connaissance d'aucune référence développant l'idée d'un ordre de comparaison autre que séquentiel.

°0°

5.4. CONCLUSION

Ce 5ème Chapitre a porté sur une application à l'identification d'objets plans en Robotique.

Une première partie a été consacrée au développement et à la justification d'une méthode originale pour l'apprentissage d'un classifieur ε-optimal et à probabilité d'erreur inférieure à une constrainte, dans le cas d'objets entièrement observés.

On a introduit ensuite une problématique d'approche pour l'identification d'objets partiellement observés, laquelle fait intervenir un problème de recherche de facteurs dans un mot. Une contribution algorithmique à ce problème a été développée. L'algorithme proposé, qui n'est pas spécifique à cette application, se rattache au reste du mémoire par l'utilisation des mêmes techniques de prétraitement d'une partie des données (ici un facteur ou un ensemble de facteurs) et leur compilation en une structure ε-optimal (une table ou un arbre) relativement à la complexité de l'algorithme.

De ce problème de "string-matching", on passera dans le chapitre suivant à un problème de "Pattern-matching" qui sera également interprété en terme de Reconnaissance de Formes par un ensemble de caractéristiques.

°0°

CHAPITRE 6

PROCESSUS DECISIONNELS OUVERTS POUR LA GENERATION DE PLANS EN ROBOTIQUE

6.1. INTRODUCTION

6.2. MODELES DE PROCESSUS DECISIONNELS OUVERTS

6.2.1. Formalisme des Systèmes de Règles de Production

6.2.2. Formalisme de la Logique des Prédicats du premier Ordre

6.3. L'UNIFICATION PAR UN PROCESSUS DECISIONNEL FERME

6.3.1. Position du problème

6.3.2. Approches communes

6.3.3. Arbre d'unification

6.3.4. Quelques extensions

6.4. INSTANCIATION DES REGLES

6.4.1. Instanciation exhaustive

6.4.2. Instanciation sélective

6.5. APPROCHE A UN PDO GUIDE PAR RECHERCHE HEURISTIQUE

6.6. CONCLUSION

°0°

6.1. INTRODUCTION

De même que le chapitre précédent n'a traité qu'une classe de problème très restreinte en Reconnaissance de Formes, dans ce chapitre on n'abordera le thème de génération de plan en robotique que sous l'angle étroit de la problématique de cette thèse : celle de l'analyse de la complexité et l'optimisation des algorithmes du type Processus Décisionnels mis en jeu.

Un système de génération de plan, non spécifique dans sa structure à un domaine d'application particulier, se présente en général comme :

1) Une base de donnée relationnelle décrivant l'état du système (univers + générateur de plan) ;

2) Un ensemble de règles de transformation d'état (correspondant, en partie, aux opérateurs-actions élémentaires du robot), et

3) un interpréteur pour l'application et l'utilisation des règles.

Ces trois éléments sont, en fait, communs à pratiquement tout système mettant en oeuvre un mécanisme d'inférence-déduction, et elles caractérisent ce que l'on désigne en général dans la littérature par Système de Règles de Production (Cf. par exemple [29,87,242]) très largement utilisées dans les Systèmes Experts [134,135,228].

On trouvera, dans de nombreuses références (Nilsson [171] , Sacerdotti [212] , Farreny [41] , Prajoux et al [194,195] par exemple) des introductions assez complètes sur les divers aspects des Systèmes de Règles de Production spécifiques à la génération de plan en robotique. On y relève que, même en se restreignant aux systèmes limités à des plans purement séquentiels, de très nombreux problèmes demeurent ouverts (interaction-consistance des règles ; caractérisation de l'état et gestion de l'espace de recherche ; backtracking ; complétude ; efficacité des générateurs ; interaction planification-exécution ; ..., etc), et que de plus en plus les ambitions des chercheurs se portent sur des

systèmes capables de générer des plans complexes : plans arborescents, plans avec parallélisme, plan avec boucles.

Une analyse immédiate des systèmes existant$ dans le sens de notre problématique, montre que dans tous les cas l'opération la plus fréquemment effectuée, la plus cruciale et pénalisante pour l'efficacité de ces systèmes, est l'opération d'unification ou "Pattern-matching", i.e., celle qui permet l'accès relationnel à la base de donnée.

Dans un cas particulier, celui de la demi-unification, on peut considérer cette opération comme un PDF, et lui appliquer les techniques d'optimisation développées en première partie de ce mémoire. On propose ici une méthode dans ce sens, qui permet de réduire considérablement le coût du "Pattern-matching". Elle est basée sur :

- la structuration de la base de donnée par l'unification aux conditions (ou prédicats) ;

- le prétraitement des conditions en un arbre d'unification ; et

- le prétraitement des antécédents des règles en un arbre d'instanciation.

Dans une première section, on introduit les Systèmes de Règles de Production comme modèle de Processus Décisionnels Ouverts, en développant essentiellement le formalisme issu des règles de réécriture, et en mentionnant quelques aspects des systèmes utilisant le formalisme de la logique des prédicats du premier ordre. On situe alors le problème de l'unification, en argumentant en faveur d'une demi-unification où l'utilisation de certains opérateurs de "matching" puissant serait contrainte. Puis, on développe la méthode proposée pour l'unification et l'instanciation des règles. On termine le chapitre par une section prospective sur les possibilités d'utilisation des algorithmes de recherches heuristiques, précédemment développés, pour guider et optimiser le Processus Décisionnel Ouvert qu'est le générateur de plan.

°0°

6.2. MODELE DE PROCESSUS DECISIONNELS OUVERTS

Le but de cette section est d'introduire des notations et concepts sur les Systèmes de Règles de Production (SRP) nécessaires au reste du chapitre. On en profitera pour proposer un modèle de Processus Décisionnels Ouverts (PDO), en parallèle avec celui précédemment introduit des PDF, en montrant ce qui distingue un SRP d'un système de règles de décision.

Un PDF porte sur un système représenté dans un espace discret fini. Dans chaque état de cet espace, une et une seule règle est valide. Le but du Processus Décisionnel est d'acquérir suffisamment d'information sur l'état du système, à un moment donné, pour déterminer quelle est cette règle valide. L'application de la règle trouvée n'intéresse plus le PDF.

Dans un PDO, on dispose également d'un ensemble fini de règles de décision, mais :

- d'une part, l'espace d'état discret du système peut être infini ;

- d'autre part,zéro ou plusieurs règles, peuvent être valides dans un état donné, et l'application d'une règle valide transforme l'état du système en un autre état.

Le but du PDO consiste à gérer ces changements d'états, en trouvant une séquence de règles qui, partant d'un état initial, amène le système à un état-objectif (décrit explicitement, ou pour lequel certaines règles sont valides).

L'antécédent d'une règle, dans un PDF, est une ebf de logique propositionnelle.

Pour modéliser un espace d'état discret infini, on admettra, dans le cas d'un PDO, une ebf de logique des prédicats du premier ordre. Un état particulier du système consistera donc en une interprétation (modèle au sens logique) sur laquelle seront évalués les antécédents des règles, i.e., en un domaine K non vide et un ensemble d'affectations qui assignent :

- à chaque constante des ebf, un élément de K ;
- à chaque symbole fonctionnel, n-adique, une application de K^n dans K ;
- à chaque prédicat, n-adique, une application de K^n dans {Vrai, Fau}.

Le conséquent d'une règle définit le changement d'état du système, en précisant quelles sont les modifications à apporter à une interprétation particulière, laquelle constitue un modèle pour l'antécédent de cette règle.

Pour résumer, un PDO se compose de :

- un ensemble de règles-opérateurs de changement d'états ;
- une base de données définissant l'état du système et constituant une interprétation pour les antécédents des règles ; et
- une procédure, appelée interpréteur, permettant de faire passer le système d'un état initial à un état objectif.

Suivant que cet interpréteur utilise ou n'utilise pas une méthode formelle de démonstration de théorèmes, le PDO reposera sur le formalisme de la logique des prédicats, ou sur celui issu des règles de réécriture (auxquels certains réservent la dénomination de SRP). On étudiera essentiellement ce dernier formalisme, en précisant brièvement quelques inconvénients du premier dans le cas de la génération de plan.

6.2.1. FORMALISME DES SYSTEMES DE REGLES DE PRODUCTION

Ce type de formalisme s'est développé comme généralisation des règles de réécriture de Post (168) sur des expressions régulières. En tant qu'outil de définition de grammaires et de langages formels, son étude est très poussée : on sait, par exemple, établir les cas d'équivalence à une machine de Turing (94) , ou à des classes d'automates particuliers (par exemple, graphes ET/OU (80)). Mais, en tant qu'outil d'inférence et de déduction, son développement est resté essentiellement empirique ; et cela malgré une utilisation intensive depuis près d'une quinzaine d'années, et l'existence de nombreux logiciels basés sur les règles de production.

En effet, à ma connaissance, il n'existe aucune caractérisation théorique générale de la puissance déductive d'un SRP (par exemple : cas de complétude, de consistance, de décidabilité). Je ne connais pas non plus de résultat sur la puissance expressive des règles de production, en comparaison notamment aux langages des prédicats du premier ordre ou du deuxième ordre, et en fonction des opérateurs et termes autorisés dans les règles. Le rapprochement des SRP et des démonstrateurs logiques des théorèmes est fréquent dans la littérature, mais l'extension rigoureuse aux premiers des propriétés connues des seconds reste à faire (voir par exemple (154) à propos de leur présentation unifiée dans (171)). La tâche dépasse les ambitions de ce chapitre.

La syntaxe et les notations que nous utiliserons seront similaires à ceux des systèmes PSG (167) , RITA (6) , ou OPS (45,57) .

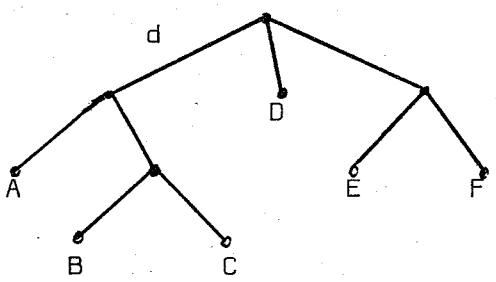
On considère trois types de symboles :

- les constantes atomiques (exemple : A, B, C, ...) ;

- les variables (exemple : x, y, z, ...) ;

- les segments (exemple : σ, ν , ξ , ...).

La base de données d'un SRP, appelée Mémoire des Faits (MF) est constituée d'un ensemble de donnée, chacune étant une liste structurée de constantes atomiques. Ainsi, par exemple, la liste d = ((A(B C)) D(E F)) est une donnée dont la structure est représentée par l'arbre :



Quelques notations sur ces arbres nous seront utiles. Chaque noeud (ou feuille) est indexé par sa position dans l'ordre scriptural (préordre des rangs x ordre transverse gauche-droite). Ainsi d(1.2.2) est la feuille étiquetée

par C, et d(3) la sous-liste (E F). On désignera par (Ø), la racine de l'arbre. De plus, si v est un noeud de l'arbre, on notera :

. L(d(v)) : le nombre de branches issues de v (exemples : L(d(1.2)) = 2, L(d(3.2)) = 0) ;

. K(d(v)) : la constante atomique étiquetant une feuille v (K(d(2)) = D, K(d(3.1)) = E).

Le plus souvent, on écrira L(v) ou K(v) en sous-entendant le nom de la donnée quand il n'y a pas de confusion.

L'ensemble des règles d'un SRP est appelé Mémoire des Règles (MR). L'antécédent de chaque règle de MR est une conjonction de prédicats, positifs ou négatifs, particuliers qu'on appellera conditions. Une condition est une liste structurée de constantes, de variables et de segments. Ainsi, par exemple, l'antécédent de la règle :

" Si [(A(B C x)) ¬(D x) (x F y)] alors ... "

comporte trois conditions, dont une négative : (D x).

Le conséquent d'une règle peut être une procédure quelconque. On ne considèrera ici que les effets du conséquent sur la MF, lesquels peuvent être ramenés aux deux opérations de suppression d'une donnée ou d'ajout d'une donnée (on parlera de donnée rentrante ou sortante en MF, du fait de l'exécution d'une règle).

La validité d'une règle repose sur une opération de "mise en correspondance" (Pattern-matching) entre les conditions de son antécédent et les données de la MF. Par définition, une condition c et une donnée d sont "en correspondance" ssi il existe une substitution qui remplace séquentiellement :

- chaque variable de c, partout où elle apparaît dans c, par une constante ou une sous-liste de constantes, et

- chaque segment de c par un ensemble ordonné de constantes et de sous-listes de constantes,

et tels qu'à l'issue de ce remplacement, c et d sont identiques.

Il s'agit, en fait, d'une opération de demi-unification particulière ("demi", car seul un des termes de l'équation c = d contient des variables ; Cf. Huet (95) pour une présentation générale), appelée dans la littérature filtrage ou appariement . On utilisera ici le terme unification (en précisant demi-unification en cas d'ambiguïté par rapport à l'unification complète), et on réservera "filtrage" à une technique particulière d'unification.

On appellera demi-unificateur de c à d la substitution, si elle existe, qui permet de les identifier ; et instance d'une variable x_i (ou d'un segment $_i$) de c, l'expression qui la remplace dans cette substitution.

Un ensemble de conditions $\{c_1, \ldots, c_k\}$ admet une unification consistante dans un ensemble de donnée ssi chaque c_i s'unifie à une donnée de l'ensemble, et tel que la même variable (ou segment) est instanciée par la même expression partout où elle apparaît dans toutes les conditions.

Une règle R dont l'antécédent comporte les conditions positives $c_1, \ldots, c_k$ et les conditions négatives $c'_1, \ldots c'_j$ est valide par rapport à un état de la MF ssi il existe une unification consistante de $\{c_1, \ldots, c_k\}$ avec les données de la MF, mais il n'existe aucune unification consistante de $\{c_1, \ldots c_k, c'_1, \ldots c'_j\}$ avec ces données. L'instance d'une règle valide est la substitution correspondante des variables de ses conditions positives. Une règle peut avoir plusieurs instances dans un état de la MF.

Ainsi, la règle de l'exemple précédent pour MF ={(A(B C I)), (A(B C J)), (D J), (I F G), (I F H)} possède deux instances :
(x —I ; y — G) et (x —I ; y — H) ; la valeur x —J n'est pas valide du fait de la condition négative (D x).

Certains systèmes permettent l'expression de contraintes sur l'instanciation des variables (ou segments) d'une condition. Par exemple (x A y | y ≠ x) autorise l'unification avec (B A C), mais l'exclut avec (B A B) (Cf. Argos 2 (41) pour la définition de nombreux opérateurs et fonctions opérant sur l'instanciation des variables).

Notons finalement, à propos de la MR, quelques remarques sur la structure des antécédents des règles :

- il peut être tenu compte d'une disjonction de conditions par la décomposition en plusieurs règles ;

- les variables (et segments) des conditions positives sont quantifiées existentiellement (par rapport à l'interprétation qu'est la MF) ;

- celles des conditions négatives (qui n'apparaissent dans aucune condition positive du même antécédent) sont quantifiées universellement ;

- il n'y a pas de symbole de prédicat constant dans une condition : elle peut avoir des variables dans n'importe quelles positions. De plus, comme un segment peut être instancié par une séquence quelconque de sous-listes, on pourra exprimer des relations quantifiant des prédicats ou des symboles fonctionnels (d'arité variable).

En plus d'une MR et d'une MF, un SRP comporte un troisième élément, l'interpréteur, qui permet l'interaction des deux premiers. Il correspond à une procédure dont le cycle de base comporte trois phases :

1 - recherche de toutes les instances de toutes les règles valides ; l'ensemble de ces instances est appelé l'ensemble de conflit ;

2 - choix d'une instance d'une règle valide (résolution du conflit) ;

3 - exécution du conséquent de la règle choisie (modification de la MF).

La contribution algorithmique de ce chapitre porte essentiellement sur la première phase. On considèrera qu'elle se décompose en deux étapes :

1.1 - unification des conditions : recherche, pour chaque condition c utilisée dans l'antécédent d'une règle de la MR, de toutes les données de MF qui s'unifient avec c ;

1.2 - instanciation des règles : détermination à partir du résultat de (1.1) de l'ensemble de conflit.

Enfin, il sera avantageux de ne faire intervenir le nom des variables (et des segments) qu'à l'étape 1.2. Aussi, pour la première étape, on considèrera que toutes les variables sont muettes, ainsi que les segments ; et deux conditions seront prises comme équivalentes si elles ne diffèrent que par le nom de leurs variables.

6.2.2. FORMALISME DE LA LOGIQUE DES PREDICATS DU PREMIER ORDRE

On peut voir essentiellement deux avantages à l'utilisation dans un PDO du formalisme des prédicats du premier ordre, l'un théorique, et l'autre pratique :

- sur le plan théorique, ce formalisme a été très extensivement étudié et on dispose des caractérisations et théorèmes généraux (consistance, complétude, semi-décidabilité, Cf. par exemple, Kleene (120)) qui font défaut au formalisme ad-hoc des SRP ;

- sur le plan pratique, on dispose d'une procédure d'inférence automatique complète (le principe de résolution (204)), de nombreuses stratégies pour la mettre en oeuvre (Cf. (22)), ainsi que de langages et d'outils informatiques intégrant cette procédure (principalement le langage Prolog (208) et ses extensions).

On trouvera, dans la littérature, plusieurs références en faveur du formalisme des prédicats du premier ordre (par exemple [55, 85, 129, 130]) et de ses nombreuses qualités. Sans prétendre remettre en cause les arguments qui y sont présentés, il me paraît cependant que les outils du type démonstrateurs de théorèmes logiques ne sont pas particulièrement bien adaptés à la génération de plans et que leurs avantages ne sont pas déterminants dans cette application. Montrons rapidement pourquoi.

Sur les trois phases de l'interpréteur d'un PDO, seule la première (et dans une moindre mesure, la deuxième) se prête à une formulation logique et à l'utilisation d'un démonstrateur de théorème. En effet, la troisième phase consiste en un changement d'état de la MF, et selon notre présentation des PDO, ce changement d'état correspond à une modification d'interprétation. Or, on ne sait pas prendre en compte, dans une procédure de preuve automatique, des interprétations variables.

Une autre façon de procéder serait de formuler les données constituant un état du PDO en tant qu'axiomes logiques. Mais, là encore, on ne saura pas conduire une inférence avec des axiomes variables (Cf. discussion dans (54) , (133)).

Deux solutions sont alors possibles pour résoudre cette difficulté sans abandonner le formalisme de la logique :

(i) on restreint l'utilisation d'une procédure de preuve formelle à la première phase de l'interpréteur, i.e., pour la validation des règles (ou la vérification de l'objectif, que l'on peut considérer comme règle particulière) ; et on se contente de procédures ad-hoc pour les autres phases de l'interpréteur (modification d'état, gestion du PDO) ;

(ii) on introduit, dans chaque prédicat, un argument supplémentaire indiquant l'état dans lequel ce prédicat est à considérer ; un changement d'état correspondra donc à une fonction de transformation de cet argument.

La première solution est celle mise en oeuvre par STRIPS (43) (ainsi que ses nombreux dérivés (42,211)). Elle conduit, de mon point de vue, à un système qui ne se distingue en rien d'un SRP avec le formalisme de la

section précédente (par exemple, la réécriture des opérateurs de STRIPS en règles d'OPS ou d'ARGOS.2 est immédiate). Les mérites (réels) d'un système tel que STRIPS ne sont pas à l'actif de l'utilisation apparente des prédicats du premier ordre, car ce formalisme ne lui apporte aucun des avantages cités, mais plutôt des inconvénients :

- on perd les théorèmes généraux, en particulier la complétude n'est plus celle garantie par le principe de résolution, mais plutôt celle qu'apporte ou n'apporte pas la procédure ad-hoc de gestion du PDO et de backtracking ;

- l'utilisation d'une procédure de preuve pour la validation des règles est très inefficace : mise en oeuvre d'un algorithme d'unification complète, et structuration rudimentaire de la MF (imposée par l'interface changement d'état-démonstrateur de théorème).

La deuxième solution est celle proposée par Green (78) , puis améliorée par Warren (dans Warplan (240) . Elle permet de conduire entièrement le PDO de génération de plan par un démonstrateur de théorème, mais au prix d'un inconvénient majeur. En effet, la description d'un changement d'état par une fonction de transformation de l'argument-état oblige à décrire explicitement, non seulement les faits qui ont été modifiés dans ce changement, mais aussi ceux qui ne l'ont pas été. On retrouve ici le fameux "frame problem", évité dans les SRP par la convention que tout ce qui n'est pas modifié par le conséquent de la règle exécutée reste inchangé.

La caractérisation d'un état par un terme de prédicats logiques interdit cette convention et oblige à préciser les "frame axioms" : règles de conservation des faits dans un changement d'état. La solution de Green est impraticable, aussi Warren propose de contourner la difficulté en décrivant les faits, non plus par des prédicats, mais par des symboles fonctionnels (pour dire, par exemple que le robot r tient l'objet x à l'instant t, au lieu du prédicat TIENT (r,x,t), on écrira VRAI (tient(r,x),t), tient étant manipulé comme une fonction). L'avantage immédiat est de pouvoir quantifier les termes correspondants, et de décrire les faits invariants dans un changement d'état par un seul axiome général.

Cette approche de WARPLAN, qui semble être une des meilleures mises en oeuvre d'un démonstrateur de théorème en génération de plan, n'est cependant pas très efficace :

- elle conduit à utiliser un nombre très réduit de prédicats (par exemple, toute règle est définie par trois instances des prédicats CAN, ADD et DELETE), ce qui entraîne le démonstrateur de théorème dans de nombreuses tentatives d'unification (complète) ;

- même si on procède à une compilation préalable des clauses , , cette compilation repose en général sur une indexation par le premier prédicat de chaque clause, et l'inconvénient précédent en annulerait les bénéfices. (On trouvera d'autres causes d'inefficacité de WARPLAN dans [240] ou [36] , par exemple).

  Notons, finalement, que la stratégie de résolution (ainsi que sa complétude) est celle du démonstrateur utilisé (incomplète dans le cas de WARPLAN sur Prolog).

°0°

6.3. L'UNIFICATION PAR UN PROCESSUS DECISIONNEL FERME

6.3.1. POSITION DU PROBLEME

Dans la très grande majorité des Systèmes de Règles de Production, l'accès associatif aux données repose sur une opération de "Pattern-matching" où un des termes ne contient que des constantes. Cela est particulièrement vrai pour l'application à la génération de plan. Même les systèmes utilisant le formalisme de la logique du premier ordre et/ou implanté dans un langage disposant de procédures d'unification complète (tels que STRIPS ou NOAH sur QA4 ou QLISP (210,213)), se restreignent, en fait, à une demi-unification.

Notons que la demi- unification ne limite pas, à priori, la puissance expressive d'un système, mais impose seulement une séparation entre les faits élémentaires et les connaissances sur ces faits. La définition de propriétés générales sur un ensemble de données, telles que les fermetures de relations transitives (par exemple, "est connexe à", "est parent de"), passe par l'écriture d'un ensemble de règles auxquelles l'interpréteur peut ou non donner un statut particulier (cas des "théorèmes" d'Argos-2 (41)).

Intervient donc un premier choix que l'on peut résumer comme suit : par rapport à l'unification complète, la demi-unification réduit la complexité des algorithmes d'accès aux données et d'instanciation des règles (proposition que l'on argumentera plus loin) ; par contre, elle conduit à introduire des règles supplémentaires qui peuvent alourdir le processus de déduction. Je ne connais pas d'étude analysant quantitativement ce compromis et permettant de trancher, même dans un seul cas particulier. Cela est vraisemblablement dû au fait que jusqu'à présent, rares sont les systèmes qui exploitent pleinement les simplifications potentielles de la demi-unification ; soit tout simplement parce que ces possibilités n'ont pas été explorées ; soit le plus souvent parce que les nombreux opérateurs de "matching", qui ont été introduits pour faciliter l'écriture des règles, remettent en cause les simplifications de la demi-unification.

On retrouve, ici, un deuxième choix tout aussi important. Si l'on se restreint à des conditions ne contenant que des variables et des constantes, alors le demi-unificateur d'une condition c et d'une donnée d, s'il existe, est unique ; de plus, il peut être trouvé en temps linéaire en le nombre total de constantes de d. Si, par contre, on s'autorise l'usage des segments (ou certains opérateurs de "matching", tels que l'appartenance), alors le nombre de demi-unificateurs distincts entre d et c peut être exponentiel en le nombre de segments de c (plus exactement en le nombre maximal de segments d'une même sous-liste de c). Ainsi, par exemple, pour :

c = (σ_1 A σ_2 A σ_3 ... σ_n A σ_{n+1}), et

d = (A A ..... A) ; avec $|d| = m$; le nombre de demi-unificateurs est en $O(n^{m-n})$.

Même la recherche d'un seul demi-unificateur, s'il existe, peut être de complexité exponentielle.

On peut rapprocher ce problème de "matching" avec segments de la demi-unification en logique d'ordre 2 (un segment correspondant à une variable fonctionnelle d'arité inconnue), lequel est un problème NP-complet (Baxter, cité dans (63)).

L'inconvénient des segments apparaît donc clairement. Leurs avantages résident dans la souplesse de formulation des conditions, la puissance expressive des règles et leur facilité de programmation. Sur ce deuxième choix non plus, je ne connais pas d'étude présentant une analyse comparative détaillée et les opinions exprimées dans la littérature sont souvent très divergentes (Cf. par exemple Rosenschein (207) , ou Hayes-Roth et al[1] (87) , et Farreny[2] (41)). On note cependant une tendance vers la simplification du "matching" dans certains systèmes (par exemple ACT1 (141) , version récente de PLASMA (191) , où l'opérateur UNPACK est restreint au seul cas où il n'introduit pas un mécanisme similaire à celui des segments).

---

(1) Selon eux :... embedding complex matching processes within a pattern matcher to allow simple descriptions of rule antecedents is very misleading".

(2) Son appréciation est : "... au vu de notre expérience dans la rédaction des règles, nous attachons du prix dans l'existence de telles facilités (opérateurs et fonctions de filtrage manipulant des segments), en vue de permettre à l'utilisateur une expression plus directe des désignations ou appels associatifs".

Remarque : Si dans les deux choix précédents, on opte pour la solution la plus complexe, alors on obtiendra un problème équivalent à celui de l'unification complète en logique d'ordre 2 , dont la décidabilité était encore un problème ouvert jusqu'à récemment (Cf. Huet (95) qui le situe par rapport à la résolution d'équations dans le monoïde libre, et Makanin, cité dans (200) , qui aurait prouvé cette résolution décidable).

Il reste, pour bien situer le problème, à montrer quelle est la part relative de l'unification dans la complexité globale de l'interpréteur d'un SRP.

D'une manière générale, l'importance essentielle de cette opération pour les systèmes mettant en oeuvre un mécanisme de déduction, fut reconnue depuis longtemps (Cf. par exemple, Robinson[1] (205)). Même en se limitant aux SRP restreints au "matching" le plus simple (demi-unification sans segment), la première phase de l'interpréteur reste de loin la plus lourde algorithmiquement et la plus coûteuse.

Ainsi, Forgy (46) constate que le premier interpréteur d'OPS y consacre les 9/10ème de son temps. Dans une étude quantitative sur le système PSG, Mc Dermott et al (155) mesurent des proportions supérieures à 98 % du temps de calcul réservé à l'instantiation des règles. Leur analyse montre que la recherche de toutes les instances de toutes les règles valides est quadratique en $O(|MF| \times |MR|)$, en admettant que l'unification d'une condition à une donnée est une opération élémentaire en temps constant. Ils soulignent donc que pour des systèmes comportant plusieurs centaines (voir des milliers) de règles, les algorithmes naïfs, tels que ceux initialement implémentés dans PSG ou OPS, sont impraticables.

Au vu de ces résultats, l'approche qui a été adoptée ici privilégie l'efficacité algorithmique de l'unification aux dépens des autres considérations. On se limitera donc à une demi-unification en essayant de tirer partie au mieux

---

(1) Il souligne : "Whatever else goes on during a deduction calculation, an enormous amount of unification computation has to be done... There is accordingly a very strong incentive to design the last possible ounce of efficiency into a unification program".

de cette limitation, et en n'autorisant des opérateurs de "matching" élaborés que dans la mesure où ils ne remettent pas en cause la complexité des algorithmes utilisés ; les segments, en particulier, ne seront admis que sous certaines restrictions.

6.3.2. APPROCHES CONNUES

Dans la littérature, on propose de remédier à l'inefficacité des procédures d'instantiation dans les SRP par, essentiellement, les trois approches suivantes :

- la méthode des partitions ;
- la méthode des filtres ;
- la méthode des réseaux.

La première est très simple ; elle se contente de réduire les données d'entrées des algorithmes classiques : on précise qu'à un moment donné, seules certaines règles sont candidates à l'unification et que leurs conditions ne peuvent être unifiées qu'avec une partie des données de la MF. Cela est obtenu en définissant des partitions, soit sur les données (par exemple (169)), soit sur les règles "Active and quiescent knowledge" de (87)), soit sur les deux ("Faits"et "Problèmes" dans Argos 2 (41)).

La méthode des filtres, ainsi que celle des réseaux, est basée sur les deux propriétés suivantes des SRP :

- l'exécution d'une règle quelconque ne modifie, en général, qu'un nombre réduit de données élémentaires, i.e., entre deux états adjacents du système, la MF a peu varié (2 à 5 changements pour une MF d'une centaine de données selon Forgy). Il s'ensuit qu'une règle qui deviendra valide, dans un cycle de l'interpréteur, était souvent proche de l'être le cycle précédent, et qu'il est avantageux d'effectuer l'instanciation des règles en "différentiel", par actualisation à chaque cycle des unifications précédemment trouvées.
- la même condition apparaît souvent dans plusieurs règles ; aussi est-il intéressant d'effectuer séparemment une procédure d'unification des conditions par rapport aux données de la MF, puis d'utiliser les unificateurs obtenus pour l'installation des règles.

La méthode des filtres a été mise en oeuvre par McDermott et al sur PSG (155) . Un filtre est un programme obtenu par prétraitement de l'ensemble des règles d'un SRP. Son rôle est de sélectionner, à chaque cycle, un sous-ensemble réduit de règles candidates à être instantiées :

$$MR : \xrightarrow{\text{filtrage}} P^+ = \{\text{règles candidates}\} : \xrightarrow{\text{Instanciation}} \text{Ensemble de conflit}$$

Sur les quatre filtres proposés dans (155) , le plus efficace fonctionne de la façon suivante :

- chaque condition apparaissant dans l'antécédent d'une règle est caractérisée par la première constante qui y apparaît (dans l'ordre de Tarry) ;

- à chaque constante correspondent :

  . une liste des conditions caractérisées par cette constante ;

  . une liste des règles contenant ces conditions ;

- à chaque règle, on associe un compteur initialisé au nombre de conditions de son antécédent.

Au début d'un cycle, on examine les données de la MF qui ont varié par rapport au cycle précédent. Pour chacune, on détermine la constante qui la caractérise et la liste des règles associées à cette constante :

- pour une donnée rentrante dans MF : on décrémente le compteur de chacune des règles de la liste ;

- pour une donnée sortante de MF : on incrémente ces compteurs.

Les règles candidates sont celles dont le compteur est à zéro.

L'analyse des auteurs cités montre que la complexité de l'algorithme d'instantiation avec filtre est au plus en $O(|MR|)$; pour les exemples traités, un gain par rapport à l'algorithme naïf dans un rapport de 10 à 15 était enregistré.

La méthode du réseau de couplage a été proposée par Hayes-Roth et Mostow (86), et principalement développée par Forgy sur le système OPS (47,48). Elle constitue une extension de l'approche précédente : les tests effectués par le filtre sont poursuivis jusqu'à l'instantiation finale des règles.

Forgy caractérise chaque condition par les longueurs de ses sous-listes et ses constantes. Ainsi, par exemple, la condition (A (B C x)) est décomposée en la séquence de valeurs : L(∅) = 2 ; L(2) = 3 ; K(1) = A ; K(2.1) = B ; K(2.2.)= C.

Les séquences associées aux diverses conditions d'un système sont "concaténées" en un réseau de tests.

Exemple 6.1. : Le réseau associé aux deux règles suivantes est donné en figure 6.1 :

R_1: "Si [(A(B C x)) (x D y)] alors ... "

R_2: "Si [(A(B C x)) (x D y) (B D y)] alors ... "

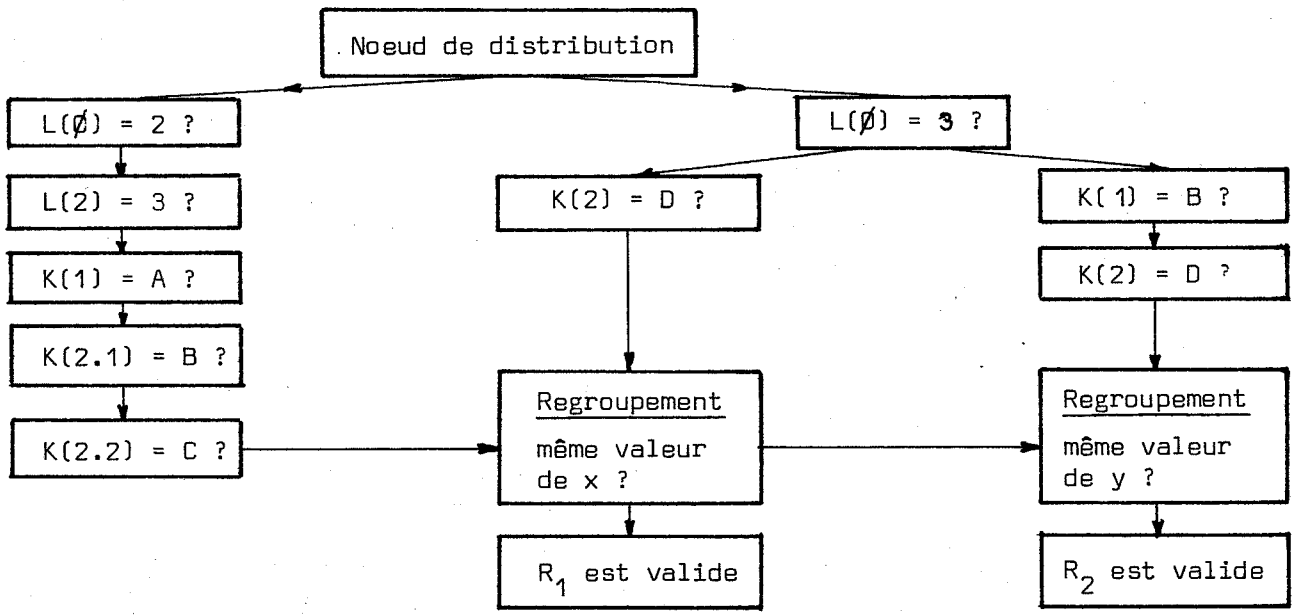


FIGURE 6.1.

Au début de chaque cycle, toute donnée rentrante ou sortante dans MF traverse le réseau sous forme de plusieurs "jetons". Si par exemple d = (B A (U V) est une telle donnée, elle sera transmise par le noeud initial à ses deux fils ; sur celui de gauche, le test "L(0) = 2 ?" ne sera pas satisfait et d ne poursuivra pas plus loin sur cette branche du réseau ; par contre, le test L(0) = 3 ? sera positif et d sera transmise aux deux fils de ce noeud,... etc. A l'issue de ces tests, la donnée arrive éventuellement en un ou plusieurs noeuds de regroupement. Chacun de ces noeuds gère une pile de produits de données et ne transmet à ses successeurs que les produits ayant même instance d'une variable.

Le parallélisme du réseau fait qu'une même donnée peut le traverser par plusieurs chemins, et donc être présente simultanément en plusieurs noeuds. Ceci présente quelques inconvénients :

- comme tous les tests sont binaires, la même information peut être extraite de la même donnée plusieurs fois (exemple : L(∅)) ;

- le même test apparaît plusieurs fois dans le réseau (exemple : K(2) = D ?) ; et plus généralement, l'information acquise dans un chemin du réseau n'est pas exploitée dans un autre chemin, ce qui introduit une grande redondance ;

- dans certains cas, il y a des problèmes de synchronisation qui font que la même instance de la même règle peut être générée plus d'une fois.

Cette approche est particulièrement intéressante si l'on peut exploiter le parallélisme du réseau par un matériel informatique multiprocesseurs spécialisé, et Forgy propose une méthode pour la conception de ce matériel (46) . Pour un processeur universel, l'approche n'est cependant pas très efficace et elle peut être améliorée.

6.3.3. ARBRE D'UNIFICATION

Si l'on admet que toute donnée de la MF s'unifie nécessairement avec au moins une des conditions d'un SRP, alors la détermination de la ou des conditions avec lesquelles s'unifie une donnée particulière peut être considérée comme un problème d'identification qu'on résoudra d'une façon avantageuse par les techniques d'arbres de décision. La méthode développée ici généralise celle des réseaux dans ce sens.

Notre hypothèse restrictive est donc qu'il n'y a pas de "données inutiles". Provisoirement, on supposera de plus que les segments ne peuvent apparaître dans les conditions que comme dernier élément des sous-listes, au maximum un par sous-liste.

Introduisons la méthode par un exemple :

Exemple 6.2. : Soit un système dont la MR contient les huit règles suivantes :

R_1 : "Si [(A (B C x)) (D x) (x F y)] alors . .."

R_2 : "Si [(A (B C x)) (D x) (x F y) (E F y)] alors ... "

R_3 : "Si [(A (B C x))¬(D x) (x F y)] alors . .."

R_4 : "Si [(A (B C x))¬(D x) (x F y) (B F y)¬(C B σ)] alors ... "

R_5 : "Si [(A (x F y)) (H x) (x F z)] alors . .."

R_6 : "Si [(A (B F σ)) (B F x)] alors . .."

R_7 : "Si [(A (B F σ)) (B F x) (x C B)] alors . .."

R_8 : "Si [(A (x B)) (x C B)] alors ..."

Ces huit règles comportent 25 conditions, mais seules les 11 suivantes sont distinctes : (A(B C x)) ; (D x) ; (x F y) ; (E F x) ; (B F x) ; (C B σ) ; (A(x F y)) ; (H x) ; (A(B F σ)) ; (A(x B)) ; (x C B).

Etant donné notre hypothèse, il suffit de discriminer parmi ces 11 conditions pour déterminer laquelle ou lesquelles s'unifient avec une donnée particulière. L'arbre suivant réalise cette discrimination (figure 6.2.).

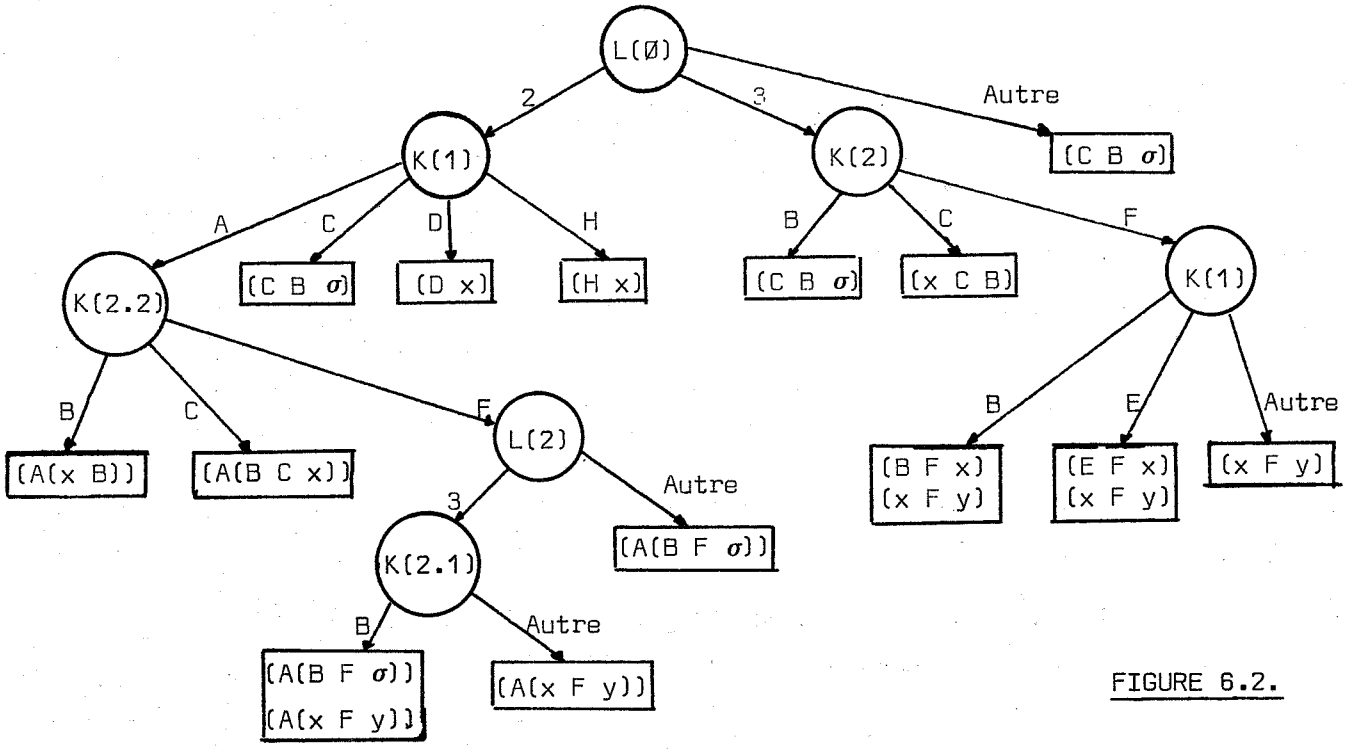


FIGURE 6.2.

Les noeuds de l'arbre d'unification sont étiquetés par des caractéristiques à évaluer sur une donnée particulière. Les étiquettes des branches correspondent aux valeurs des caractéristiques ; une branche "Autre" est associée à toutes les valeurs possibles autres que celles des branches du même noeud.

Au début d'un cycle, si d = (E F(U V W)) est par exemple une donnée rentrante ou sortante en MF, elle parcourra l'arbre de la racine à une feuille. En la racine, on détermine que d comporte trois sous-éléments. La branche correspondante mène au noeud "K (2)" dont la valeur sur d est la constante "F". Le noeud suivant extrait le premier sous-élément de d. Sa branche "E" conduit à une feuille étiquetée par les deux conditions (E F x) et (x F y), les seules à s'unifier avec d.

La procédure de génération d'un tel arbre comporte les trois phases suivantes :

1) Définition des caractéristiques discriminantes pour l'ensemble des conditions du système ;

2) Définition d'un ensemble de règles d'unification ; et

3) Génération d'un arbre de décision représentant l'ensemble de règles précédentes.

La première phase passe d'abord par l'énumération de l'ensemble des conditions distinctes du système (toutes les variables sont muettes et indifférenciées, de même que tous les segments). On examine ensuite chaque condition c_j, et on détermine toutes les caractéristiques d'un certain type définies pour une donnée s'unifiant avec c_j.

Plusieurs types de caractéristiques sont utilisables. On se contentera ici des deux types :

. L : longueur d'une liste ou d'une sous-liste (L(v) = 0 si v correspond à un atome) ;

. K : constante atomique en une certaine position (K(v) = "LISTE", ou tout autre atome réservé, si v correspond à une sous-liste).

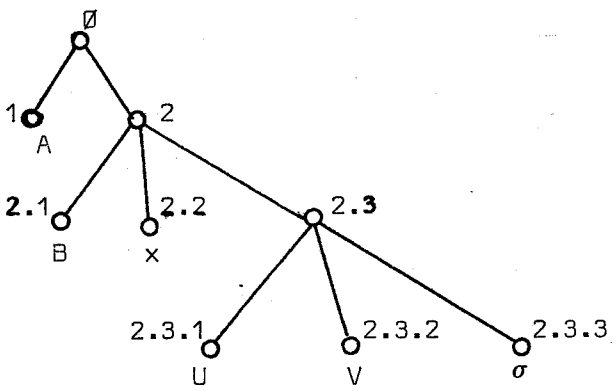
Pour énumérer les caractéristiques du type "L" ou "K" associées à une condition, il suffit de construire son arbre-structure :

- L est définie sur tous les noeuds internes et toutes les feuilles de cet arbre ;

- K est définie sur toutes ses feuilles.

Exemple 6.3 : pour **c** = (A(**B** x (U V σ))),
on aura les caractéristiques suivantes :

L(Ø), L(1), L(2), L(2.1), L(2.2), L(2.3), L(2.3.1), L(2.3.2), L(2.3.3) ;
K(1), K(2.1), K(2.2), K(2.3.1), K(2.3.2), K(2.3.3).



La phase 2 devra générer des règles d'unification du type :

"Si pour une donnée d:(L(v_1) = l_1) et (L(v_2) = l_2) ... et (K(v_i) = k_i) et ...;
alors d s'unifie avec les conditions $\mathbf{c}_{j1}$ et $\mathbf{c}_{j2}$ ..."

Du fait de la présence éventuelle de segments en fin de sous-liste, on pourra également avoir des règles avec relations d'inégalité sur les caractéristiques L :

"Si pour d:(L(v_1) $\geq l_1$) et (L(v_2) $\geq l_2$) ... alors d s'unifie ..."

On définira ces règles en construisant tout d'abord une table regroupant les valeurs de toutes les caractéristiques précédemment obtenues pour toutes les conditions du système. L'élément courant de cette table (condition C_j ; caractéristique L(v) ou K(v)) appartient à l'ensemble :

$N \cup \{ \varphi, \theta \} \cup \{ "a^+" \mid a \in N \} \cup$ {constantes atomiques du système} $\cup \{$"LISTE" $\}$

Il sera défini comme suit :

- si v désigne un noeud n'appartenant pas à l'arbre-structure de $\mathbf{c}_j$, alors K(v) = θ , L(v) = θ ;
- si v est un noeud de $\mathbf{c}_j$, on a alors un des cas suivants :
  - . v est une feuille étiquetée par une variable : K(v) = L(v) = φ
  - . v est une feuille étiquetée par un segment : K(v) = L(v) = θ
  - . v est un noeud dont un fils est étiqueté par un segment : L(v) = a^+, avec a = (nombre de fils de v)-1
  - . dans tous les autres cas, l'élément courant de la table est égal à la valeur de L(v) ou K(v) évaluée sur la condition $\mathbf{c}_j$.

Exemple 6.4. : la table correspondant aux conditions de l'exemple 6.2. est :

	(A(B C x))	(A(x F y))	(A(B F σ))	(A(x B))	(D x)	(H x)	(B F x)	(x C B)	(E F x)	(C B σ)	(x F y)
L(0)	2	2	2	2	2	2	3	3	3	2^+	3
L(1)	0	0	0	0	0	0	0	φ	0	0	φ
K(1)	A	**A**	**A**	A	D	H	B	φ	E	C	φ
L(2)	**3**	**3**	2^+	2	φ	φ	0	0	0	0	0
K(2)	LISTE	LISTE	LISTE	LISTE	φ	φ	F	C	F	B	F
L(3)	θ	θ	θ	θ	θ	θ	φ	0	φ	θ	φ
K(3)	θ	θ	θ	θ	θ	θ	φ	B	φ	θ	φ
L(2.1)	0	φ	0	φ	θ	θ	θ	θ	θ	θ	θ
K(2.1)	B	φ	B	φ	θ	θ	θ	θ	θ	θ	θ
L(2.2)	0	0	0	0	θ	θ	θ	θ	θ	θ	θ
K(2.2)	C	F	F	B	θ	θ	θ	θ	θ	θ	θ
L(2.3)	φ	φ	θ	θ	θ	θ	θ	θ	θ	θ	θ
K(2.3)	φ	φ	θ	θ	θ	θ	θ	θ	θ	θ	θ

les symboles " θ " et " φ " sont utilisés avec les mêmes définitions que précédemment (chapitres 1 et 3) :

-" φ " : valeur indéterminée ou variable (exemple : L(2.1) pour (A (x B)), x pourra être instancié par une constante, ou une sous-liste de la longueur quelconque)

-" θ " : caractéristique non définie (exemple : L(3) pour (A (B C x)) , aucune donnée s'unifiant avec cette condition ne peut comporter 3 sous-éléments). Il suffit d'une seule donnée pour laquelle la caractéristique est non définie pour imposer le symbole θ (exemple : L(3) pour (C B σ), ou K (2.1) pour (D x)).

Le symbole " a+" correspond à une relation d'inégalité pour caractéristique du type L : si v désigne une sous-liste de c_i de longueur (1+a) et comportant un segment, alors pour toute donnée s'unifiant avec c_i on aura : L (v) $\geq$ a ; le cas d'égalité correspond à une instanciation du segment à la séquence vide (exemples : L(o) $\geq$ 2 pour (C B σ), ou L(2.1) $\geq$ 2 pour(A (B F σ))) . Si la ligne associée à L(v) comporte les nombres k_1, ... k_i et les symboles "a_1^+",...,"a_j^+", le domaine d'application de L sera inclus comme étant :
$\{m, m+1, \ldots, M\}$ avec $m = \min_{i,j}\{k_i ; a_j\}$ et $M = \max_{i,j}\{k_i ; a_j - 1\}$.

Par convention, on ajoutera une valeur "Autre" au domaine d'application de toute caractéristique comportant les symboles " φ " ou "a^+" dans sa ligne. Certaines caractéristiques dont le domaine d'application est réduit à un singleton (L (2.2), L(2.3) et K(2.3) pour l'exemple 6.4) seront supprimées de la table car elles ne contribuent en rien à la discrimination entre les conditions.

A toute colonne de la table correspond une règle d'unification particulière, dont l'antécédent comporte les relations d'égalité ou d'inégalité sur toutes les caractéristiques de la table, autre que celle dont la valeur est " θ " ou " φ " dans cette colonne. Par exemple, la régle correspondant à la dernière colonne de (6.4) est :
" si pour une donnée d on a (L (ϕ) = 3) et (L (2) = O) et (K (2) = F) alors d s'unifie avec (x Fy) ". L'ensemble de régles ainsi obtenu ne constitue pas un système (X, A, D, R) consistant et complètement spécifié. On s'y ramènera en procédant d'une manière similaire à celle du chapitre 5. On remarque cependant que :

- la table précédente est une généralisation du type "décision grid chart" : plusieurs règles peuvent être simultanément valides ; elles correspondront à des données s'unifiant à plusieurs conditions ;

- l'introduction des valeurs "Autres" rend le système complet ;

- les relations de dépendance entre caractéristique sont implicites : à l'interprétation des symboles "φ" et "a^+" près , les seules combinaisons de valeurs de caractéristiques possibles sont celles explicitées dans la table.

On n'insistera pas davantage sur le passage à une représentation normalisée de ce système de règles d'unification. En effet, une telle représentation n'est ici nécessaire que pour pouvoir effectuer des regroupements de termes sur $(\bigvee_i Z_i)$, calculer les quantités q_i, et évaluer les heuristiques guidant la recherche d'un arbre d'unification optimal ou ϵ- optimal. Or ces calculs nécessitent la connaissance d'une distribution de probabilité sur $(\bigvee_i Z_i)$, ce qui semble très difficile à acquérir dans cette application. Concrétement, celà exige d'avoir a priori la liste de toutes les données susceptibles d'être produites dans le système et de connaître la fréquence d'apparition et de disparition de chacune d'entre elles en MF. Par ailleurs, admettre l'hypothèse d'équiprobabilité des états de $(\bigvee_i Z_i)$ n'est pas plus réaliste que de supposer l'information précédente disponible.

Il nous semble donc plus raisonnable dans cette application d'abandonner l'exigence d'un critère d'optimisation exacte et de procéder heuristiquement à la recherche d'un "bon" arbre d'unification.

Informellement, une procédure (similaire à A1) générant un arbre d'unification à partir de la table précédente pourraît être la suivante :

- Parmi les caractéristiques partout définies (n'ayant pas "φ" dans leur ligne), en choisir une X_i comme racine de l'arbre ;

- Associer à cette racine autant de branches qu'il y a de valeurs dans le domaine d'application de X_i (y compris la valeur "Autre", s'il y a lieu), et faire correspondre à chaque branche une sous-table de la table initiale contenant :

  - l'ensemble des conditions suivantes :

    . pour une branche étiquetée par une constante K : toutes les conditions pour lesquelles X_i vaut K, ou "φ", ou " a^+ " si k est une constante numérique avec $a \geqslant k$;

    . pour une branche étiquetée par "Autre" : toutes les conditions pour lesquelles X_i vaut "φ" ou "a^+", pour tout a.

  - l'ensemble des caractéristiques de la table initiale sauf X_i, en modifiant les valeurs "θ" ou "φ" s'il y a lieu pour certaines caractéristiques (Exemples :-1) pour la branche $L(\emptyset) = 3$, les caractéristiques L(3) et K(3) passent de "θ" à "φ" pour (C B σ) ;
    -2) pour K(1) = B, L(1) passe de "φ" à 0 sur (x F y)); puis en éliminant les caractéristiques non discriminantes.

- Associer à une branche dont la sous-table comporte un ensemble vide de caractéristique, une feuille étiquetée par toutes les conditions de cette sous-table (toute donnée traversant l'arbre jusqu'à cette feuille s'unifie nécessairement avec toutes ces conditions) ;

- Associer à toute autre branche le sous-arbre défini récursivement par la même procédure.

Proposition 6.1. : Cette procédure génére un arbre d'unification correct pour l'ensemble des conditions du système.
Preuve : Il suffit de remarquer que l'ensemble des caractéristiques de la table initiale est complet : la reformulation de toutes les conditions du système à partir de cette table est immédiate. Par ailleurs, les caractéristiques non définies et les contraintes de succession qu'elles entraînent, ne limitent

pas la réalisabilité du système : pour toute L(v) ou K(v) non définie sur certaines conditions,il existe L(w), w : liste incluant v, qui est définie et permet d'isoler ces conditions. Le reste de l'argument découle sans difficulté de nos hypothèses : absence de données inutiles, et segment uniquement en fin de sous-listes. □

Le choix de la caractéristique à mettre en un noeud donné de l'arbre d'unification sera effectué en fonction d'heuristiques estimant :

1 - La quantité d'information apportée par une caractéristique (son "pouvoir discriminant"). On reviendra à l'idée de partition fine et équilibrée. En fait, plutôt qu'une partition, chaque caractéristique en un noeud de l'arbre réalise un recouvrement de l'ensemble des conditions de la sous-table à laquelle ce noeud est associé (ce n'est une partition que si le noeud n'a pas de branche "Autre"). Aussi on pourra favoriser la caractéristique:
(i) dont le nombre de "φ" ou de "a^+" dans sa ligne est minimal, ou celle
(ii) dont le nombre moyen de conditions par branche est minimal, ou bien celle
(iii) - dont le nombre maximal de condition par branche est minimal.
(Pour les 5 caractéristiques partout définies dans l'exemple précédent ces 3 critères donnent :

	L(∅)	L(1)	K(1)	L(2)	K(2)
(i)	1	2	2	3	2
(ii)	4.33	6.5	3.28	4.75	3.8
(iii)	7	11	6	7	6

)

Le 1er entier ne prend pas en compte le nombre de branches d'un noeud. Dans le 2ème, la notion d'"équilibre d'une partition" est masquée par l'opération de moyenne. Le 3ème nous paraît être plus intéressant. En cas d'ex aequo, on favorise la caractéristique ayant le nombre maximum de branches.

2- Le coût d'évaluation d'une caractéristique. Dans tous les cas il y aura :

(i) - un coût d'accès à la sous-liste v ;

(ii)- un coût de calcul de K(v) ou L (v) ;

(iii)- un coût de détermination de la branche correspondante à la valeur trouvée.

La quantification de ces 3 composantes dépend bien entendu de chaque implémentation particulière. Dans celle effectuée par R. Sobek (227) à laquelle nous avons contribué , la structure de données est celle du système hôte (Interlisp). Le coût d'accès à v est donc une fonction du rang de v dans l'arbre-structure; et de sa position dans l'ordre d'alignement transverse (pour v = (i.j.k), l'accès à v se fera par $CAD^{k-1}\ AD^{j-1}\ AD^{i-1}$ R, en admettant des fonctions CAR, CDR généralisées). De plus, dans cette implémentation, chaque noeud de l'arbre est une "Hash-table". Une caractéristique de type K nécessite une opération pour trouver K(v) dans la table s'il y est, et sinon un test supplémentaire pour déterminer si K(v) est une liste. Le coût d'une caractéristique de type L est proportionnel à la longueur de la liste v. Les deux notions de coûts conditionnels et de coûts variables sont donc présentes ici. L'absence d'un critère d'optimisation exacte fait qu'on se contentera, en pratique, de coûts constants.

Exemple 6.5:Pour les conditions de l'exemple 6.2, ces heuristiques conduisent à l'arbre suivant (figure 6.3).

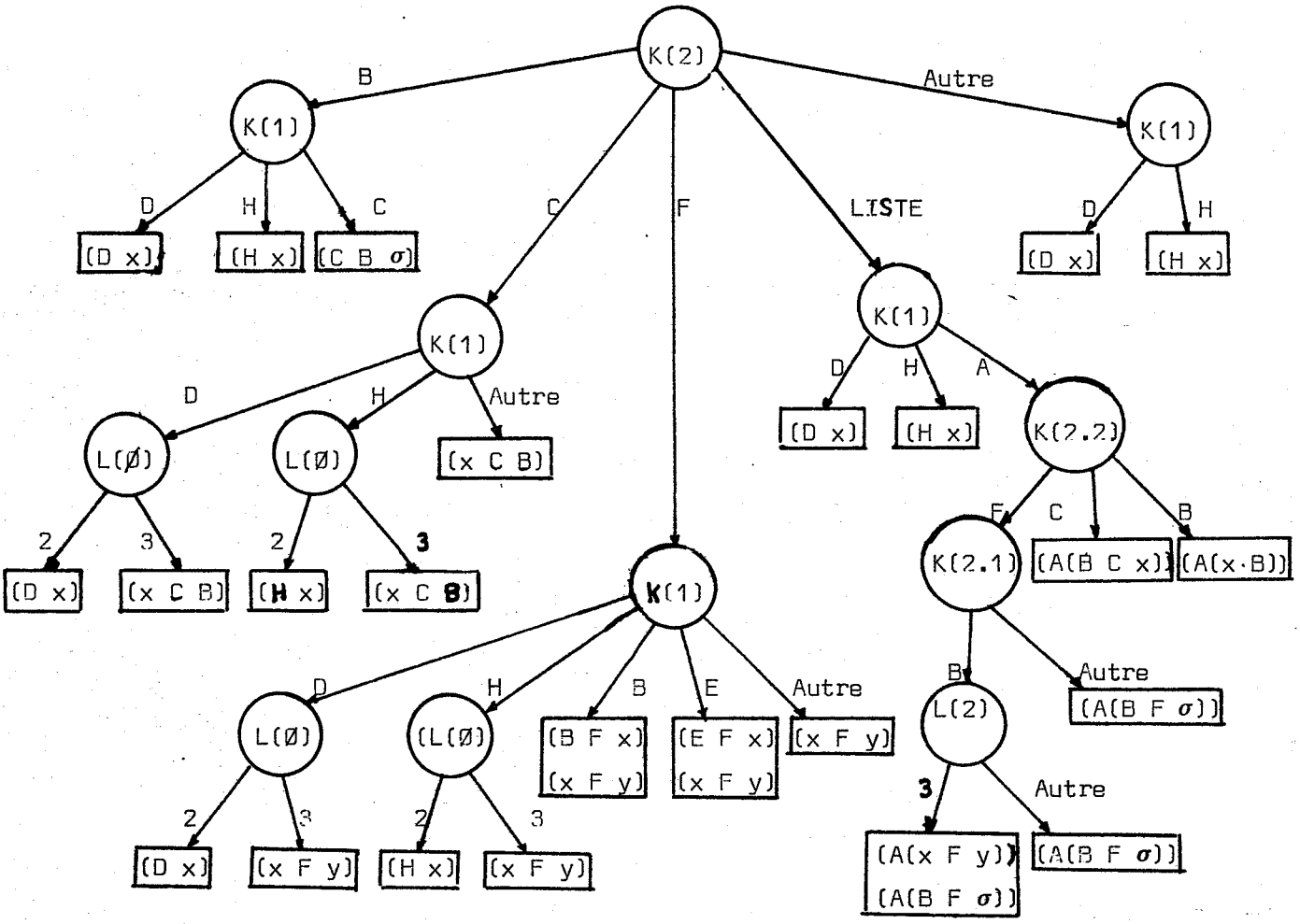


FIGURE 6.3.

6.3.4. QUELQUES EXTENSIONS

On analysera, dans cette section, la relaxation des hypothèses restrictives faites précédemment, ainsi que les problèmes de mise à jour de l'arbre d'unification en cas de modification d'un système de règles.

On pourrait craindre que la transformation d'un ensemble de conditions en un arbre d'unification ne détruise une des principales qualités des systèmes de règles de production : leur souplesse ou "granularité". Cette situation est comparable à la compilation d'un code par rapport à son interprétation : initialement, on fait appel à l'interpréteur (un programme de pattern-matching classique), puis on procède à la compilation après une première phase de mise au point. Il est néanmoins intéressant de prévoir la possibilité de modifications peu fréquentes dans un système de règles de production tout au long de son utilisation.

On a proposé, en section 4.4.2., un algorithme de mise à jour d'un arbre de décision qui suppose que l'ensemble des caractéristiques du système demeure inchangé. Ici, les caractéristiques évoluent avec les modifications des conditions, aussi on procèdera d'une manière légèrement différente :

- si c_j est une condition qui vient d'être supprimée du système : on l'enlève de toutes les feuilles de l'arbre où elle apparaît, on supprime toutes les branches menant à une feuille vide et tous les noeuds ayant un fils unique ;

- si c_j vient d'être ajouté au système : on parcourt récursivement l'arbre à partir de la racine de la manière suivante :

  . en une feuille : on détermine une table d'unification pour c_j et l'ensemble des conditions étiquetant cette feuille, et pour toutes leurs caractéristiques (autres que celles dans le chemin menant à la feuille), et on génère un sous-arbre représentant cette table qui remplacera la feuille ;

  . en un noeud étiqueté par une caractéristique X_i, on cherche à évaluer X_i sur c_j :

* si X_i n'est pas définie : on procède, comme pour une feuille, par rapport à toutes les conditions issues de ce noeud ;

* si X_i = k et qu'il existe en ce noeud une branche étiquetée par la constante k : on poursuit le parcours de l'arbre le long de cette branche ;

* si X_i = k et qu'il n'existe en ce noeud ni branche k, ni branche "Autre", alors on ajoute au noeud une branche étiquetée k menant à une feuille contenant c_i ;

* si X_i = k, et qu'il y a une branche Autre mais pas de branche k, alors on ajoute une branche k à laquelle on associe un sous-arbre représentant c_i et toutes les conditions de la branche "Autre" ;

* si X_i = "φ" ou "a^+" et qu'il existe une branche "Autre", alors on poursuit le parcours de l'arbre sur toutes les branches du noeud pour X_i = "φ", ou sur toutes les branches autres que celles étiquetées par $k < a$, pour X_i = "a^+" ;

* si X_i = "φ" ou "a^+" et qu'il n'existe pas de branche "Autre", alors on ajoute une branche Autre menant à une feuille contenant c_i, et on poursuit le parcours de l'arbre sur les autres branches comme précédemment.

Analysons, à présent, l'hypothèse relative aux segments. L'usage d'un outil tel que l'arbre d'unification est lié au fait qu'entre une donnée et une condition, le demi-unificateur, s'il existe, est unique. La méthode précédente reste applicable, quelle que soit la position d'un segment unique dans une sous-liste, et cela à condition de pouvoir accéder aux éléments de cette sous-liste aussi bien à partir du début, qu'à partir de la fin. L'accès ascendant dans une liste peut être fait de deux manières différentes :

- en parcourant la liste jusqu'à la fin (NIL) et en reprenant en sens inverse à partir de là (coûteux en temps de traitement) ;

- en utilisant des structures de listes doublement pointées (coûteux en espace mémoire).

On notera algébriquement $v = (j_1 . j_2 . \ldots j_k)$; $j_i \in \mathbb{Z}^*$, le sous-élément de rang k d'une liste, obtenu en prenant :

- si $j_k > 0$: le $j_k^{ème}$ élément de la sous-liste $(j_1 . \ldots j_{k-1})$;
- si $j_k < 0$: le $(L(j_1 . \ldots j_{k-1}) + 1 - j_k)^{ème}$ élément de $(j_1 \ldots j_{k-1})$.

On suppose, dans tous les cas, que v est accessible et que les caractéristiques L(v) et K(v) sont évaluables si définies (on parlera de caractéristiques négatives si v comporte un indice négatif).

La génération de l'arbre d'unification se fait d'une manière similaire à précédemment. Soulignons rapidement quelques points spécifiques à cette extension :

- définition de la liste de l'ensemble des caractéristiques évaluables : une sous-liste de rang k peut être adressée de 2^k façons différentes, ce qui peut conduire à 2^k caractéristiques identiques du type L ou K. Aussi, on restreindra l'adressage négatif aux sous-listes contenant un segment ailleurs qu'en dernière position (pour accéder aux éléments situés après le segment) ;

- définition de la table d'unification : on conservera toutes les caractéristiques négatives de la liste précédente, car même celles non discriminantes dans la table initiale peuvent le devenir dans une sous-table ;

- génération de l'arbre d'unification : on évite l'évaluation de caractéristiques identiques le long d'un même chemin par :

  . le mécanisme des contraintes de succession (exemple : la caractéristique K(2) n'est pas définie pour $(\sigma\ H)$, à moins que L(Ø) n'ait été préalablement évaluée) ; et

  . l'élimination des caractéristiques identiques dans une sous-table (exemple : si L(Ø) = 3, alors L(1) ≡ L(-3), K(2) ≡ K(-2), ...).

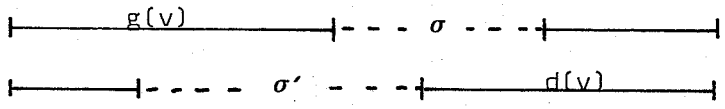
Par ailleurs, le domaine d'application des caractéristiques L sera défini différemment : pour v donné, sur l'ensemble des sous-listes v des conditions contenant un segment , on détermine :

g(v) = nombre maximum de sous-éléments de v à gauche du segment ;

d(v) = nombre maximum de sous-éléments de v à droite du segment ;

le domaine de L(v) sera $\{m, m+1, \ldots, g(v) + d(v) - 1\}$.

La valeur "Autre" correspond alors à $L(v) \geq d(v) + g(v)$; auquel cas, pour aucune paire de sous-listes v il n'y a recouvrement entre les éléments adressés négativement (partie droite), et ceux adressés positivement (partie gauche d'un segment). On aura le schéma suivant :



Exemple 6.6 : Pour l'ensemble de huit conditions suivantes (A(σ F x)) ; (A(B Fσ)) ; (A(x B)) ; (D x) ; (σ H) ; (x C B) ; (C σ B) ; (A(B C x)), le compilateur décrit dans [227] fournit, en 0.6 s, l'arbre d'unification de la figure 6.4 . On y remarque, par exemple, que le chemin contenant les deux caractéristiques K(2.-2) et K(2.2) correspond à une longueur $L(2) \geq 4$.

L'information apportée par une caractéristique peut être estimée par les mêmes heuristiques que précédemment. On remarque, cependant, qu'à moins de disposer de listes doublement pointées, le coût d'évaluation d'une caractéristique négative est plus élevé : il intègre le coût d'accès au dernier élément d'une sous-liste et de retour jusqu'à la position adressée.

De plus, l'introduction de segments dans des positions quelconques accroit, d'une manière importante, les recouvrements des sous-listes ; ce qui nécessite un plus grand nombre de tests pour discriminer entre les conditions. A titre simplement illustratif, l'arbre de la figure 6.2 représente onze conditions ne contenant des segments qu'en fin de sous-liste, et celui ci-dessus n'en représente que huit ; cependant :

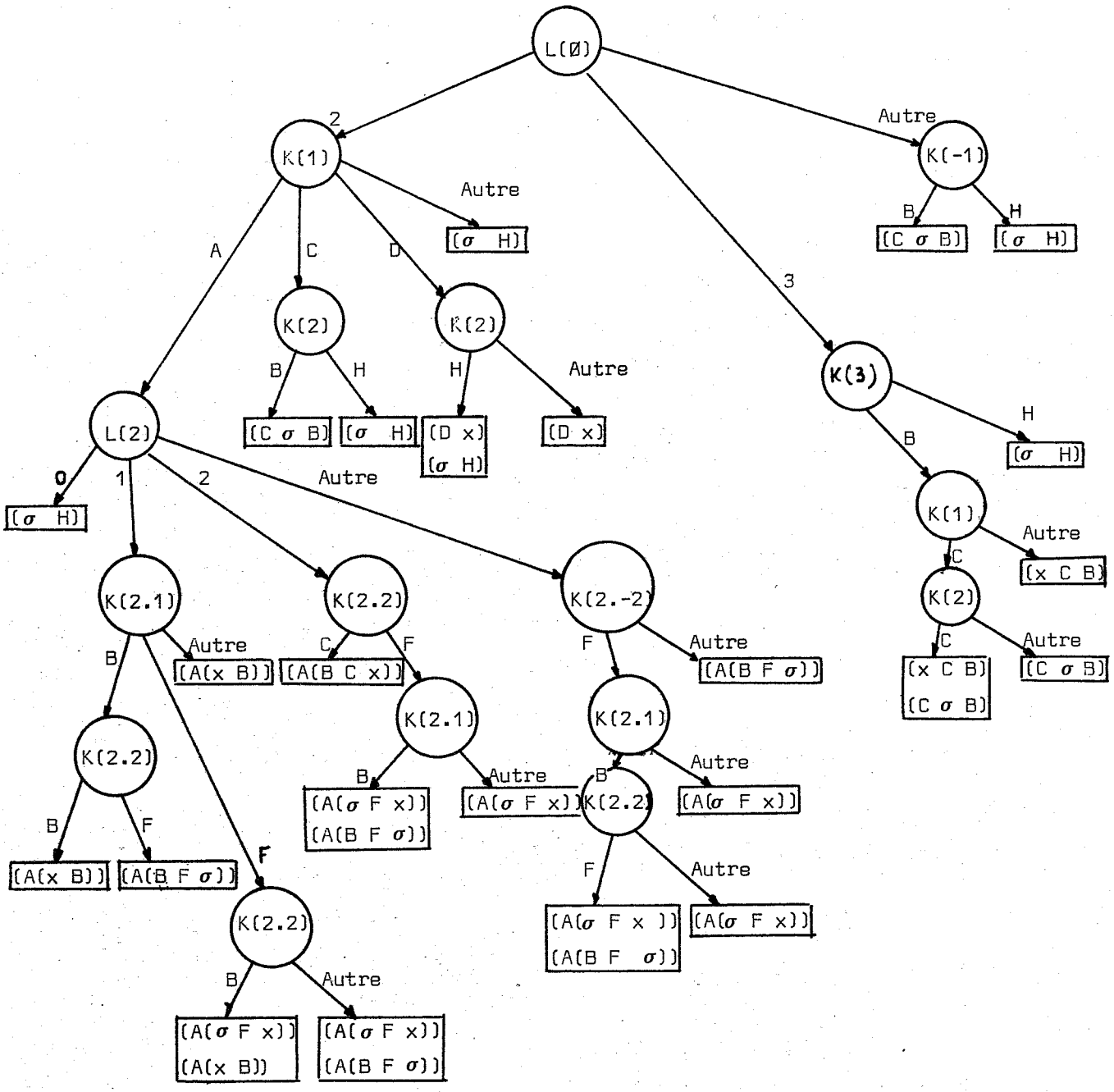

L(Ø)
2
K(1)
Autre
(σ H)
A
C
D
K(2)
K(2)
B
H
H
Autre
(C σ B)
(σ H)
(D x)
(σ H)
(D x)
L(2)
0
1
2
Autre
(σ H)
K(2.1)
K(2.2)
K(2.-2)
B
Autre
(A(x B))
C
F
(A(B C x))
F
Autre
(A(B F σ))
K(2.2)
K(2.1)
K(2.1)
B
F
(A(x B))
(A(B F σ))
F
K(2.2)
B
Autre
(A(σ F x))
(A(x B))
(A(σ F x))
(A(B F σ))
B
Autre
(A(σ F x))
(A(B F σ))
B
Autre
(A(σ F x))
K(2.2)
F
Autre
(A(σ F x))
(A(B F σ))
(A(σ F x))
3
Autre
K(-1)
B
H
(C σ B)
(σ H)
K(3)
B
H
(σ H)
K(1)
Autre
(x C B)
C
K(2)
C
Autre
(x C B)
(C σ B)
(C σ B)

FIGURE 6.4.

- le premier comporte 14 feuilles de hauteur moyenne 2.86 ;
- le deuxième en comporte 24, de hauteur moyenne 3.88, soit un test de plus pour chaque feuille.

On retiendra donc que malgré la compilation en arbre, l'unification avec segments reste un processus complexe.

Abordons maintenant l'hypothèse d'absence de données ne s'unifiant avec aucune condition. Son utilité pour notre approche est évidente : elle permet de résoudre un problème de demi-unification coûteux en le transformant en un problème de discrimination très simple.

Sa justification est davantage discutable. Dans la plupart des systèmes de règles de production, les données de la MF sont produites de façon interne, par les différentes règles exécutées. La partie conséquent d'une règle peut ajouter une donnée dans la MF, la supprimer si elle s'y trouve déjà, ou la modifier. Ces deux dernières opérations portent, en général, sur les données qui ont instancié l'antécédent de la règle, alors que l'ajout d'une donnée correspond à un appel associatif d'un sous-ensemble de règles, et dans tous ces cas, l'hypothèse est justifiée.

Par ailleurs, une opération sur une donnée ne s'unifiant avec aucune condition alourdit inutilement le système : on cherchera, sans succès à l'unifier, on abandonnera sa suppression s'il s'agit de la supprimer ou de la modifier, ou bien elle sera ajoutée autant de fois que la règle sera exécutée. Il est donc souhaitable d'épurer le conséquent des règles de ce type d'opération, mais il n'est pas très réaliste d'en faire porter la responsabilité sur le programmeur.

Dans certains cas, il sera possible de vérifier a priori notre hypothèse sur les données des conséquents de toutes les règles du système. Mais, souvent, ces données comporteront des variables quantifiées qui n'acquèreront une valeur qu'à l'instanciation de la règle, et la vérification de l'hypothèse sera très difficile (elle dépend de l'état initial et de tout état obtenu à partir de lui par fermeture transitive de l'application des règles).

Par ailleurs, dans l'application à la génération de plan en robotique, il est très important de pouvoir prendre en compte des données générées de façon externe par un des systèmes de perception du robot. Aussi bien pour ces données, que pour celles dans l'état initial de la MF, notre hypothèse n'est pas justifiée.

Cependant l'approche développée jusqu'ici restera utilisable, au prix d'un supplément de coût, dans tous les cas où des données "inutiles" risquent d'être rencontrées. A certaines feuilles de l'arbre d'unification précédemment obtenu, on ajoutera des tests de confirmation. Ainsi, par exemple, l'arbre de la figure 6.2. sera transformé en (figure 6.5).

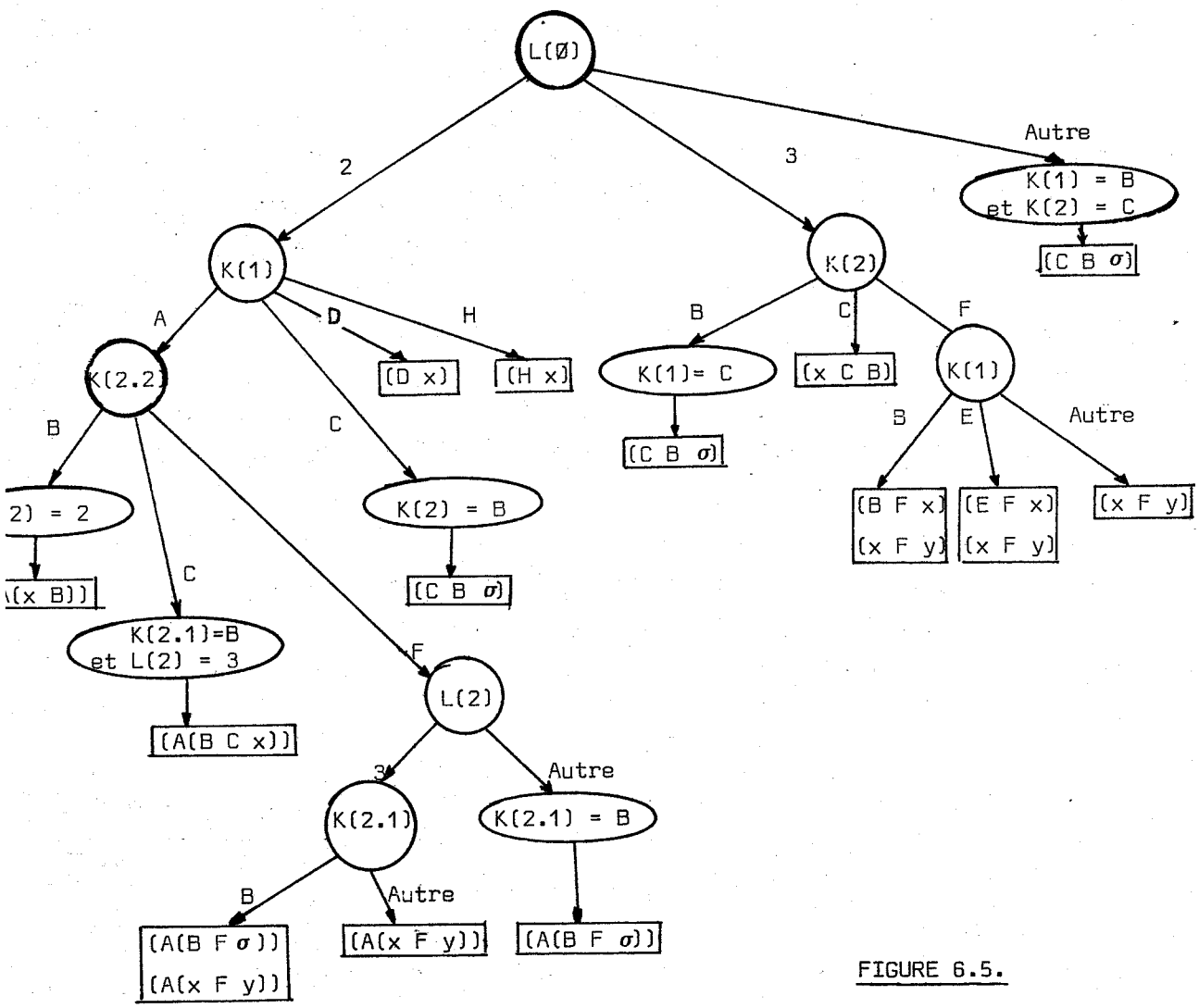


FIGURE 6.5.

Pour une donnée traversant l'arbre :

- si elle rencontre en un noeud interne une caractéristique non définie, ou bien si le test associé à un noeud de confirmation est négatif, alors cette donnée sera ignorée (on ne cherchera pas à la supprimer, à l'ajouter ou à la modifier dans la MF) ;

- sinon, elle sera unifiée aux conditions de la feuille atteinte et traitée normalement.

On remarque que l'arbre d'unification ainsi augmenté restera, malgré tout, beaucoup plus efficace que la procédure de matching classique (le test de confirmation se fera pour une seule condition, ou un groupe de conditions indistinguables), et ne contiendra pas de test redondant comme le réseau d'unification.

Par ailleurs, il sera toujours possible de traiter séparément, par deux arbres d'unification, l'un augmenté, l'autre pas, les données produites de façon externes, et celles pour lesquelles l'hypothèse est vérifiée.

Terminons cette section en envisageant très rapidement les possibilités d'extension de l'arbre de demi-unification à une unification complète (sans segment).

Une donnée d contenant des variables ne traversera plus l'arbre par un chemin unique, mais éventuellement par plusieurs chemins : si (d(v) est une variable, on poursuivra sur toutes les branches issues d'un noeud contenant la caractéristique K(v). De plus, en certains noeuds, on rencontrera des caractéristiques non définies pour d : L(w) ou K(w), w sous-liste de v, ne seront pas définies si d(v) est variable. Là encore, il faudra prendre toutes les branches issues de ces noeuds.

Finalement, les feuilles qui seront atteintes pour d ne correspondront pas aux conditions s'unifiant effectivement avec d, mais à celles susceptibles de le faire. Il restera à vérifier, pour chaque paire condition-donnée, que la correspondance variables-termes obtenue est bien un unificateur valide (substitution acyclique, sans affectations multiples à la même variable).

En effet, l'arbre ne prend pas en compte le nom des variables d'une condition c, mais uniquement de leur position dans c. Le fait qu'une donnée d traverse l'arbre (avec tests de confirmation) jusqu'à une feuille étiquetée par c, signifie simplement que :

- les structures de c et de d se correspondent (i.e., seraient identiques si on remplace les variables par des sous-arbres) ; et

- chaque constante atomique de c correspond à la même constante dans d, ou a une variable, ou bien la sous-liste dans laquelle figure cette constante correspond à une variable.

Admettons, par exemple, que la donnée d = (x(B F z)) traverse l'arbre de la figure 6. (en faisant abstraction des conditions contenant des segments). Elle aboutira aux feuilles étiquetées par les conditions (A(x F y)), (D x), et (H x) dont aucune ne s'unifie à d (par rapport à la première, on aura la correspondance (A/x, B/x, y/z), où x apparaît 2 fois associée à 2 termes distincts).

On peut conclure ces remarques en notant que :

- si les variables qui apparaissent dans les données de la MF et celles qui figurent dans les conditions appartiennent à deux ensembles disjoints, alors l'unification pourra se faire par un arbre ;

- sinon, dans le cas général, l'arbre d'unification apportera :

  . une restriction de l'ensemble des conditions auxquelles l'unification pourra être tentée ; et

  . éventuellement, une simplification des algorithmes d'unification qui seront alors utilisés, l'opération se ramenant à une simple vérification d'un système de multi-équations très simplifié (dans le sens de Martelli et Montanari [150], en notant que l'exploitation de leur algorithme dans cette direction serait un problème intéressant).

- dans tous les cas, le parcours de l'arbre sera plus coûteux que pour la demi-unification : tests en chaque noeud plus complexes, et exploration d'un plus grand nombre de chemins.

6.4. INSTANCIATION DES REGLES

Nous avons décomposé la première phase de l'interpréteur en deux étapes :

- unification des conditions aux données de la MF ;
- instanciation des règles.

L'arbre d'unification fournit, en permanence, la liste de toutes les données s'unifiant à chaque condition. Au début de chaque cycle de l'interpréteur, il fournit de plus une mise à jour de cette liste : données qui viennent de la quitter et celles qui viennent de s'y ajouter.

A partir de là, nous aborderons la composition de ces données en instance de règles valides. Deux cas seront considérés :

- celui où tout l'ensemble de conflit est requis, i.e., toutes les instances de toutes les règles valides ; et
- celui où l'on cherche une instance d'une règle particulière à partir d'information a priori sur l'utilité des règles dans un état.

6.4.1. INSTANCIATION EXHAUSTIVE

Soit une règle R dont l'antécédent comporte les conditions c_1, c_2, ... c_k ; et soit C_i l'ensemble de toutes les données de la MF s'unifiant à un moment donné avec la condition c_i. Chaque élément de C_i est un demi-unificateur entre une donnée et c_i, i.e., une substitution qui à chaque variable ou segment de c_i fait correspondre une sous-liste d'atomes, ou une séquence de sous-listes.

On cherche, pour la règle R, tous les k-tuples du produit cartésien $C_1 \times C_2 \times \ldots C_k$, tels que les substitutions associées à chaque k-tuple soient consistantes, i.e., qu'à la même variable ou segment soit substituée, partout où elle apparait dans l'antécédent de R, la même expression.

Bien entendu, à partir de là, les variables ne sont plus considérées comme muettes. On fait également intervenir, à ce niveau, les contraintes et fonctions de "matching" (telles que, par exemple : (x A y | P(x,y)) qui contraint, pour une donnée s'unifiant à (x A y), les instances de x et de y à vérifier le prédicat P).

Le problème est donc de faire,le plus efficacement possible, les produits Cartésiens relatifs aux antécédents de toutes les règles du système ; et de vérifier sur chaque k-tuple, la consistance de (et, éventuellement, certaines contraintes sur) l'instanciation des variables.

Ce problème est bien connu dans le domaine des Bases de données relationnelles, et correspond à l'opération de "joint". Si r est une relation à n attributs incluse dans $D_1 \times \ldots \times D_n$, et r' une relation à m attributs incluse dans $D'_1 \times \ldots \times D'_m$, chaque D_i étant un ensemble de données élémentaires ; alors le "joint" de r et de r', $r * r'$ est le produit Cartésien $r \times r'$ restreint aux éléments ayant des valeurs identiques sur le même attribut (Cf. par exemple (236)).
Une des voies poursuivie par les chercheurs dans ce domaine et qui semble être très fructueuse, est la conception de matériels spécialisés (adjoints à des processeurs universels) pour la réalisation de cette opération "joint" (158).

Algorithmiquement, l'approche qui me paraît la plus efficacement adaptée aux spécificités d'un système de règles de production est celle de Forgy (46) . Il détermine l'ensemble de conflit en différentiel par rapport à l'état précédent, en supprimant les instances qui ne sont plus valides et en ajoutant celles qui le sont devenues. Ceci est effectué par un réseau d'instanciation comportant deux types d'opérateurs : les noeuds "Produit" pour les conditions positives et les noeuds "Validation" pour les conditions négatives. Chaque noeud

gère un produit Cartésien restreint, sur lequel consistance et contraintes d'instanciation des variables ont été vérifiées. Le réseau est construit de façon à ce que les sous-produits de conditions communes aux antécédents de plusieurs règles ne soient effectués qu'une seule fois.

On propose dans (69) quelques améliorations à cette approche, qui consistent essentiellement en :

- un troisième type d'opérateur, dit de "Partition", destiné à regrouper davantage les sous-produits communs à plusieurs antécédents, et

- une procédure d'actualisation de l'ensemble de conflit, réduisant les parcours des données sortantes et rentrantes dans le réseau.

Exemple 6.7. : Pour présenter informellement cette méthode, reprenons les huit règles de l'exemple 6.2. On peut remarquer, à propos de leur antécédent, que :

- la variable z apparaît une seule fois dans R_5, de même que x dans R_6, y dans R_1 et R_3, ou σ dans R_4, R_6, R_7. L'instanciation de ces variables n'est donc pas contrainte : elles n'interviendront pas dans le réseau d'instanciation pour les règles mentionnées, et le traitement des conditions où elles apparaissent en sera simplifié ;

- les trois conditions (A(B C x)), (D x) et (x F y) sont communes aux quatre règles R_1, R_2, R_3, R_4, de plus, aucune paire de condition de ces quatre règles ne figure dans une autre règle. On peut donc traiter l'instanciation de ces quatre règles indépendamment des autres. On le fera par le réseau suivant (figure 6.6) :

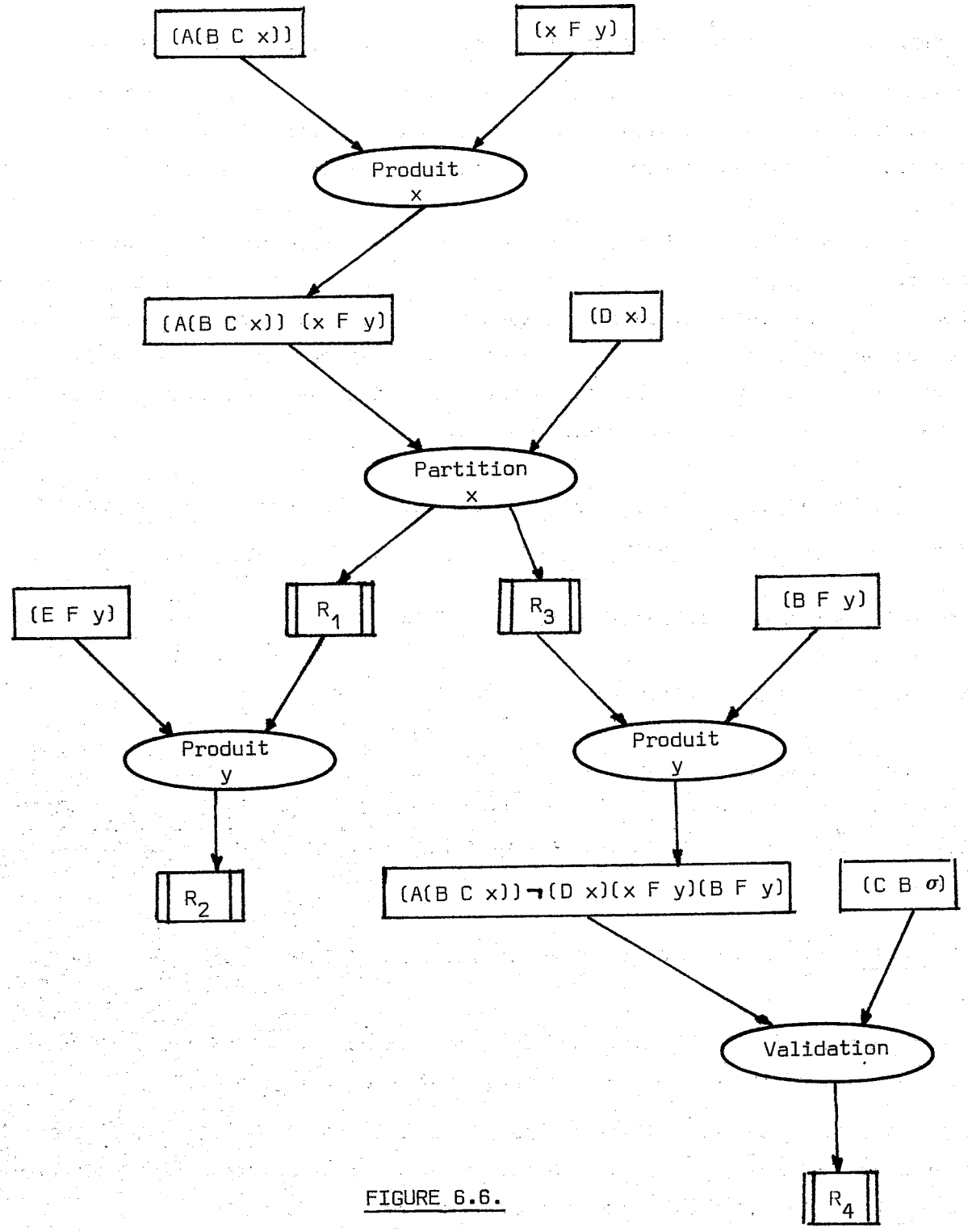

(A(B C x))
(x F y)
Produit x
(A(B C x)) (x F y)
(D x)
Partition x
(E F y)
R1
R3
(B F y)
Produit y
Produit y
R2
(A(B C x))¬(D x)(x F y)(B F y)
(C B σ)
Validation
R4

FIGURE 6.6.

Il s'agit d'un arbre structuré par un préordre de rang et ayant :

- un ensemble de noeuds correspondant à des piles de k-tuples de données, et
- un ensemble de noeuds correspondant à des opérateurs particuliers ayant chacun deux noeuds pères et un ou plusieurs fils auxquels ils sont chargés de transmettre les modifications intervenues dans les données de noeuds pères.

On rappelle que la MF est structurée par l'unification aux conditions du système. Pour chaque condition c_i, on dispose de la liste des données qui s'unifient à c_i, et des modifications à apporter à cette liste au début de chaque cycle : données rentrantes ou sortantes en MF (information gérée par une pile de pointeurs aux données, ou plus économiquement, de pointeurs aux instances des variables de c_i correspondante à chaque donnée).

Ces modifications se propagent dans l'arbre d'instanciation en deux phases :

- traitement des données sortantes, puis
- traitement des données rentrantes.

La propagation de chaque phase se fait d'une façon complète par rang (on actualise tous les noeuds donnés de rang r avant d'aborder la propagation au rang r+1). Les procédures relatives à chacun des noeuds-opérateurs sont les suivantes :

Noeud "Produit" : possède un fils unique et ses deux pères jouent un rôle symétrique :

- pour une donnée d (ou un k-tuple) sortante dans un père : on examine la pile-fils, et tout produit où figure d est marqué comme produit sortant ;
- pour une donnée d rentrante dans un père : on examine la pile de l'autre père, et pour chaque d' tel que (d x d') correspond à une instanciation consistante, on transmet au fils (d x d') comme produit rentrant.

Noeud "Partition" : possède deux pères, l'un au moins correspondant à une condition négative, et deux à trois fils (dans le cas général, les pères sont associés à deux conditions c_1 et c_2 et les fils aux piles des trois produits $(c_1 \times c_2)$, $(c_1 \times \neg c_2)$, $(\neg c_1 \times c_2)$).

- pour une donnée d sortante en c_1 : on passe en revue la pile $(c_1 \times c_2)$, tout produit (d x d') où apparaît d est marqué sortant de $(c_1 \times c_2)$, et si d' ne figure qu'une seule fois dans $(c_1 \times c_2)$, d' sera marquée rentrante dans $(\neg c_1 \times c_2)$;

- pour une donnée d rentrante en c_1 : on examine c_2, tout produit (d x d') consistant est marqué rentrant dans $(c_1 \times c_2)$ et s'il n'existe aucun tel d' dans c_2, d est marqué rentrant dans $(c_1 \times \neg c_2)$.

Noeud "Validation" : correspond à un cas particulier du noeud "Produit" où seul un des deux fils $(c_1 \times \neg c_2)$ ou $(\neg c_1 \times c_2)$ est retenu (on maintient cependant la pile $(c_1 \times c_2)$).

Remarques :

- un produit tel que $(\neg c_1 \times \neg c_2 \times c_3)$ peut être obtenu comme fils d'un noeud "Partition" (ou "Validation") dont un père serait (c_1) et l'autre $(\neg c_2 \times c_3)$ par exemple ;

- chaque opérateur est étiqueté par la, ou les variables sur lesquelles doivent porter les vérifications de l'instanciation. Pour une condition ne contenant aucune variable contrainte (cas de (B F x) dans R_6), les opérations précédentes se ramènent au plus à un produit Cartésien, et au mieux, à une transmission ou blocage d'une pile entière.

L'implémentation de cette phase d'instanciation des règles, par la méthode décrite ici, n'a pas encore été faite. Aussi, nous ne proposons pas de procédure particulière pour la génération de l'arbre d'instanciation. On pense pouvoir s'inspirer des algorithmes utilisés en Programmation dynamique "non-série" (problème primaire d'ordre d'élimination des variables (15)) ; ou bien

de certaines méthodes d'optimisation utilisées en compilation, telles que les techniques de décomposition et de recherche de sous-expressions communes à plusieurs expressions (5) . Il nous reste également à préciser plusieurs détails sur la réalisation des opérateurs proposés.

6.4.2. INSTANCIATION SELECTIVE

Dans cette section, ainsi que dans la suivante, aucune contribution spécifique ne sera développée. On se contentera de proposer quelques idées prospectives sur des problèmes ouverts.

S'il est exact que peu de données seront en moyenne modifiées dans la MF à chaque cycle, la phase d'unification données-condition sera peu coûteuse. Sa complexité est proportionnelle au nombre de données modifiées et à la hauteur moyenne de l'arbre, laquelle, raisonnablement, est en O(log $|C|$); $|C|$ étant le nombre de conditions distinctes du système .

La phase d'instanciation des règles risque, par contre, d'être plus pénalisante, puisqu'elle porte non seulement sur les données modifiées, mais aussi celles qui ne l'ont pas été.

Sa complexité fera intervenir le nombre d'instances de règles modifiées dans l'ensemble de conflit, ainsi que le produit des nombres totaux de données s'unifiant aux conditions de ces règles ; ce qu'on peut estimer en $O(|MR| \times (|MF| / |C|)^a)$, en admettant a conditions par règles et $|MF| / |C|$ données dans la pile associée à chaque condition.

Par ailleurs, une fois l'ensemble de toutes les règles valides généré, il faudra résoudre le conflit en choisissant une instance particulière, lequel choix peut lui-même être très coûteux et de complexité proportionnelle à la dimension de l'ensemble de conflit.

L'idéal serait de pouvoir anticiper sur la résolution du conflit avant l'instanciation des règles. En quelque sorte, on admettrait par avance que les

règles "utiles" dans un état sont susceptibles d'y être valides, et on fournirait à la procédure d'instanciation le nom de la règle qui aurait été sélectionnée dans ce cas, en lui demandant d'en chercher une instance (éventuellement, une instance particulière). Connaissant les données s'unifiant à chaque condition, la procédure d'instanciation serait alors immédiate. Notons qu'on pourrait également fournir à cette procédure un sous-ensemble, ordonné par préférence, de règles dont il s'agirait de trouver la première valide.

Le principe auquel on pense immédiatement pour la mise en oeuvre d'une telle approche, est celui du chaînage arrière selon l'archétype canonique de STRIPS.

L'intégration des outils présentés ici se fera de la manière suivante :

- l'objectif à atteindre (sous forme de conjonction de conditions positives ou négatives) est considéré comme une règle particulière dont le conséquent produit l'arrêt du système ;

- toutes les conditions des antécédents des règles (y compris l'objectif) sont compilées en un arbre d'unification (l'arbre-antécédent) ;

- les données dans les parties "ajout" des conséquents des règles ("add-list") contiennent, en général, des variables qui sont instanciées en même temps que la règle . On les considère comme un ensemble de conditions, et on les compile en un deuxième arbre d'unification (l'arbre-ajout). Les feuilles de cet arbre seront étiquetées, non seulement par ces conditions, mais aussi par les règles qui les produisent. D'une manière similaire, on construit un troisième arbre (l'arbre-suppression) pour les "delete-list" des conséquents.

On procèdera alors ainsi :

- toutes les données de la MF définissent l'état initial traversant l'arbre-antécédent ;
- on met la règle-objectif au sommet d'une pile de règles en attente d'instanciation, puis on itère sur ce qui suit jusqu'à la rencontre d'une règle d'arrêt :

  . on cherche à instancier la règle au sommet de la pile (produit consistant des sous-ensembles de données associées à son antécédent) :

    - si on y parvient, alors on exécute son conséquent (modification de l'état de la MF et incidence sur l'arbre-antécédent), on l'enlève de la pile, et on poursuit sur la règle suivante ;
    - sinon :

  . on choisit un sous-ensemble Δ de données, positives ou négatives, suffisantes pour compléter une instanciation de cette règle que l'on met au sommet de la pile ;

  . les données de Δ traversent, suivant leur type, l'arbre-ajout ou l'arbre-suppression, et on choisit une règle, parmi celles associées aux feuilles atteintes par ces données, pour la mettre au sommet de la pile.

Cette procédure admet l'hypothèse implicite que le choix de la règle à instancier pourra se faire sur la base de la simple unification des données nécessaires à l'objectif partiel (ensemble Δ) avec celles des données modifiées par cette règle, chaque couple étant considéré isolément (on ne cherchera pas une instance complète d'un conséquent).

Si cette hypothèse n'est pas valide, alors on n'élimine plus le problème de l'instanciation exhaustive. On le déplace simplement de l'antécédent des règles à leur conséquent, ce qui correspond tout de même à un gain, car l'ensemble de conflit est, en général, beaucoup plus important que l'ensemble des règles modifiant certaines données.

Soulignons, finalement, que les trois arbres de demi-unification proposés ici amélioreront l'efficacité des générateurs de plan du type STRIPS, mais ne résoudront en rien tous les problèmes qui leur sont inhérents, en particulier ceux des divers choix non-déterministes (quel ensemble Δ, quelle règle), et ceux de gestion du graphe de recherche et de backtracking.

6.5.APPROCHE POUR UN PDO GUIDE PAR UNE RECHERCHE HEURISTIQUE

Les méthodes de recherche heuristique dans un graphe d'espace d'état (ou dans un graphe ET/OU) sont parmi les outils les mieux formalisés de l'Intelligence Artificielle, et considérés dans la littérature comme le moyen le plus adéquat pour guider un PDO. Pourtant, à ma connaissance, aucun SRP n'utilise véritablement un algorithme de recherche heuristique (à l'exception d'exemples académiques simples, tels que le 8-puzzle). On se contente, le plus souvent, d'une simple recherche en profondeur, limitée arbitrairement, et sans possibilité de détection de circuit.

Deux arguments pour justifier le recours à des méthodes dont on connaît, a priori, l'inefficacité :

- la difficulté de définir des heuristiques suffisamment précises et ayant les propriétés requises de monotonie ou minoration ; et surtout

- la difficulté de caractériser l'état u courant de l'espace de recherche et de répondre très rapidement aux questions (fondamentales pour tous les algorithmes du chapitre 2) : u est-il développé ?,u est-il pendant ?, u est-il fils de v ? ($u \in Q$?, $u \in P$?, $u \in \Gamma(v)$?).

Si l'on admet que l'optimisation (ou l'ε-optimisation) du coût de mise en oeuvre d'un plan (selon un modèle précis différenciant les coûts de chaque opérateur) est, aujourd'hui, un problème secondaire par rapport à l'optimisation de son coût d'obtention (i.e., l'optimisation du PDO qui le génère), alors la première difficulté peut être en partie résolue par les heuristiques connues du type "Means-end-analysis" (40) et ses nombreuses extensions (65,89) . En effet, le modèle de coût associé à l'espace de recherche est alors très simple, et seule une heuristique du type h_c est nécessaire.

Un problème beaucoup plus complexe est à l'origine de la deuxième difficulté. Un état u de l'espace de recherche est caractérisé par toutes les données présentes en MF (de l'ordre de la centaine de données structurées), et répondre aux questions évoquées, signifie la sauvegarde de tous les états antérieurs de la MF, et la comparaison de l'état u à tous ces états.

La complexité de la tâche semble, a priori, remettre en cause le bénéfice de tout algorithme de recherche efficace (à l'exception, encore une fois, de problèmes de jeux et autres cas simples où u peut être caractérisé par un entier tenant sur un mot machine). Elle nous semble, cependant, envisageable grâce à la structuration de la MF proposée ici, et à l'utilisation des techniques les plus efficaces de tri. Développons rapidement quelques idées dans ce sens.

On remarque, tout d'abord, que seules les données s'unifiant avec au moins une condition du système, partitipent à la caractérisation de l'état u (les autres données n'interviennent en rien dans l'instanciation des règles dans u, ou dans tout autre état). De plus, l'arbre de demi-unification est construit de telle sorte que toute donnée "utile" débouche sur une et une seule feuille de cet arbre. La structuration de la MF pas sous-ensembles de données associées aux feuilles de l'arbre, est donc complète et sans redondance.

Par ailleurs, les données dans chacun de ces sous-ensembles ne diffèrent que par les instances des variables. On ne retiendra donc que ces instances, en considérant celles de chaque variable comme une "clé" sur un domaine ordonné (par exemple : ordre alphabétique, ou position dans la table des symboles des constantes correspondantes).

A la $i^{ème}$ feuille de l'arbre de demi-unification sera donc associé un sous-ensemble E_i de r-tuples (r = nombre de variables pour cette feuille) de clés ordonné, ainsi que certaines caractéristiques de ce sous-ensemble faciles à déterminer (par exemple, son cardinal $|E_i|$).

Finalement, il sera avantageux de caractériser chaque état en <u>différentiel</u> par rapport à l'état initial u_o. Pour un arbre de demi-unification possédant l feuilles, u_o sera caractérisé par l sous-ensembles vides. Si $(E_1,\ldots E_l)$ caractérisent un état v, alors la caractérisation de u fils de v est $(E'_1,\ldots,E'_l)$, chaque E'_i étant obtenue par la mise à jour de E_i (ajout des données rentrantes et suppressions des données sortantes produites par la règle faisant passer de v à u et aboutissant à la $i^{ème}$ feuille). Si E_i n'a pas varié de v à u, il sera <u>commun</u> aux deux états.

On associe à chaque ensemble d'états particuliers (états pendants, états développés) un arbre de recherche (par exemple du type "Quad-tree", ou toute autre technique de structuration d'un ensemble de l-tuples (123)) dont les feuilles sont étiquetées par les ($|E_1| \cdot |E_2| \cdot \ldots \cdot |E_1|$) distincts des états de cet ensemble. Pour déterminer si $u \in P$, on commence par chercher si la configuration ($|E_1| \cdot \ldots |E_1|$) qui le caractérise est présente dans cet ensemble (recherche très peu coûteuse) :

- si elle n'y est pas, alors $u \notin P$, et on l'ajoute éventuellement (si u doit figurer dans P) ;

- sinon, on restreint la comparaison de u aux états ayant le même l-tuple ($|E_1| \cdot \ldots |E_1|$) que u. Pour chaque paire u-v à comparer, on passera en revue, dans un certain ordre, les feuilles distinctes qui leur correspondent. Les ensembles de r-tuples $E_i(u')$ et $E_i(v)$ étant ordonnés, leur comparaison se fera séquentiellement (premier élément à premier élément, ...) en s'arrêtant et en abandonnant la comparaison à v dès la première différence.

En première analyse, la lourdeur de l'approche proviendrait essentiellement de la gestion des structures associées aux sous-ensembles d'états, et de l'ordonnancement des l-tuples d'ensembles (E_1, ..., E_1) caractérisant chaque état. Elle nous paraît cependant praticable, surtout si le nombre de données modifiées par chaque règle est faible, et si l est grand.

Insistons sur le fait que le bénéfice éventuel d'une telle gestion de l'espace de recherche ne devra pas être apprécié uniquement au niveau du coût (utilisation de meilleurs algorithmes pour guider le PDO, simplicité du backtracking), mais également au niveau décisionnel, puisqu'elle permet de préserver la semi-décidabilité du système : s'il existe une solution, elle sera trouvée (preuve d'admissibilité ou d' ϵ-admissibilité des algorithmes du chapitre 2).

°0°

6.6. CONCLUSION

Ce Chapitre a permis de développer un modèle de Processus Décisionnels Ouverts, pour la génération de plans en Robotique, à partir des Systèmes de Règles de Production.

L'analyse des PDO, dans l'optique de cette thèse, montre que la complexité d'une étape élémentaire de ces processus provient essentiellement de l'accès associatif aux données.

Le Chapitre a développé une méthode originale, qui interprète cet accès comme un PDF, et qui compile en conséquence les antécédents des règles. Ce pré-traitement conduit à une procédure de demi-unification améliorant les "Pattern-matcher" connus.

La complexité de l'ensemble du PDO est celle des algorithmes et stratégies de recherche mis en oeuvre. Une approche pour l'utilisation des algorithmes de recherche heuristique $\mathcal{E}$-admissibles développés au Chapitre 2 est proposé.

°0°

CONCLUSION

Les contributions que propose ce mémoire ont été résumées dès l'introduction et rappelées à la fin de chacun des chapitres où le problème correspondant est développé. Je n'y reviendrai pas ici. Un aspect mérite cependant d'être souligné, c'est les rapprochements et parallèles qui sont établis entre des concepts et des outils utilisés jusqu'à présent dans des disciplines relativement cloisonnées, particulièrement :

- entre les méthodes de Recherche Heuristique et Résolution de Problèmes en Intelligence Artificielle, et celles d'Optimisation Combinatoire en Recherche Opérationnelle, et
- entre les systèmes de règles mettant en oeuvre des Processus Décisionnels Fermés, et ceux conduisant à des Processus Décisionnels Ouverts.

Ces rapprochements se sont révélés fructueux à plusieurs niveaux :

- extension des procédures de Recherche Heuristique à des schémas d'approximation, et leur application à l'optimisation des PDF,
- mise en évidence d'un PDO comme processus faisant intervenir une séquence de PDF, et optimisation de chacune de ses étapes élémentaires en conséquence,
- formulation de l'optimisation d'un PDO en tant qui minimisation de la complexité d'une procédure de Recherche Heuristique.

A côté des contributions et résultats obtenus, que j'ai essayé de dégager tout le long de ce mémoire, l'approche adoptée et les solutions proposées se situent dans des limites qu'il me paraît nécessaire de relever dans cette conclusion.

La limitation la plus importante me semble être la restriction de texte la problématique abordée à des processus essentiellement séquentiels, et la non-prise en compte dans ces processus d'un quelconque niveau de parallélisme. Le risque réel (dont je suis conscient pour l'avoir observé sur des problématiques étroites telle que la minimisation du nombre de "portes" d'un opérateur Booleen), est qu'une technologie, qui se maintient en progression géométrique, ne rende obsolète une approche qui n'intègre pas les problèmes de décomposition de processus décisionnel en plusieurs processeurs/décideurs et ne prend pas en compte leur coordination. Dans le domaine de la Robotique, ce risque n'est plus à mettre au conditionnel. Deux remarques cependant permettent

d'espérer que les solutions et outils développés ici ne seront pas totalement inutiles dans une approche plus générale :

1) Sauf dans des cas très heureux et des situations de grande abondance (processus entièrement décomposable, disposant d'un processeur par tâche élémentaire et convergent en une seule étape) un processus décisionnel restera séquentiel à deux niveaux :
- au niveau de chacun des processus élémentaires qui le décomposent, et
- au niveau de leur coordination, la décomposition restant en général imparfaite.

2) Le degré de parallélisme le plus ttendu ne changera en rien la classe de complexité d'un problème NP-complet ; et des algorithmes exponentiels, tels que ceux abordés au Chapitre 2, continueront à exiger des approches par approximation, des heuristiques et des stratégies de résolution.

La deuxième ouverture importante que cette thèse n'a pas abordée est celle de l'algorithmique probabiliste. Le problème de l'optimisation dynamique (qui me paraît très séduisant, et pour lequel une approche a été proposée au Chapitre 4) ne pourra en particulier être résolu d'une façon satisfaisante sans l'introduction de modèles théoriques et de mesures d'incertitude ; modèles et mesures nécessaires pour quantifier finement, aussi bien le degré d'approximation des solutions fournies (au lieu des simples majorants proposés ici), que la complexité des algorithmes de recherche.

Les difficultés de ce dernier problème, pour des algorithmes tel que A_ε et AO_ε , sont nombreuses et ont été soulignées aux Chapitres 2 et 4. Il est cependant important de s'y atteler car l'approche expérimentale n'est pas très satisfaisante à mon avis. La validation empirique d'un algorithme ou d'une méthode de résolution, ou plus généralement l'assimilation d'un programme à une expérience scientifique (assimilation courante en Intelligence Artificielle) fait face à une difficulté méthodologique importante ;
- d'une part le nombre d'observations et de points de mesures qui peuvent être fait dans le meilleur des cas restera très négligeable par rapport aux dimensions de l'espace des données raisonnablement possibles (même les 30000 essais de (64) sur le domaine très restreint du "8-puzzle" représentent moins de 1 échantillon sur 3 million de points), et - d'autre part le degré de variabilité du comportement d'un programme, d'un jeu de données à l'autre, est très grand.

Au delà de ces 2 limitations majeures de l'approche développée dans ce mémoire, plusieurs problèmes ouverts ont été rencontrés, et pour lesquels je n'ai pu proposer au mieux que la définition d'une problématique et une esquisse possible de résolution. Parmi ces problèmes, il me paraît intéressant de rappeler :

- l'application à l'identification d'objets partiellement observés, problème à mi-chemin entre la perception bidimensionnelle et tridimensionnelle,
- la mise en oeuvre effective d'algorithme tel que A_ε et AO_ε pour guider et optimiser un PDO, et l'évaluation du coût de gestion de la recherche inhérent à ces algorithmes,
- l'extension des méthodes de demi-unification (dans le sens proposé ici ou différemment) à l'unification complète, et la comparaison aux meilleurs algorithmes connus,
- l'étude comparative des variantes d'accès associatifs (unification demi-unification, avec ou sans opérateur du type segment) en quantifiant le compromis entre la complexité du matching et la puissance expressive des règles,
- dans le même sens que précédemment, étude comparative (plus approfondie que les parallèles syntaxiques proposées ici) entre les systèmes de règles de ré-écriture et ceux mettant en oeuvre une procédure d'inférence formelle.

Pour clore ce mémoire, je voudrais mentionner que sur plusieurs des problèmes ouverts abordés dans cette conclusion (notamment celui des processus décisionnels non strictement séquentiels, et celui de l'identification d'objets partiellement observés) le travail de recherche a déjà démarré au L.A.A.S. J'espère que je pourrai continuer à m'associer à cette recherche, et que sur un plan plus général, la lecture de ce mémoire contribuera à une meilleure formulation de ces problèmes ouverts et à leur résolution.

°0°

ANNEXE

SEGMENTATION D'UN ESPACE DE REPRESENTATION

On discutera, dans cette annexe, deux méthodes simples de segmentation d'un espace de représentation constitué par n caractéristiques conditionnellement indépendantes. On établira les deux propositions :

Proposition 5.1. : Le partitionnement par seuil unique entre deux classes ne permet pas, en général, de respecter une contrainte sur la probabilité d'erreur .

Proposition 5.2. : L'introduction d'une bi-partition par classe (sous-domaine de sélection et sous-domaine d'exclusion pour chaque classe ω_j) permettra de respecter cette contrainte, mais ne garantira pas une identification de l'objet inconnu à la classe la plus probable (compte-tenu de l'information disponible).

Segmentation par frontière unique

La preuve de (5.1) est extrêmement simple à établir sur un contre-exemple comportant une caractéristique unique X et deux classes ω_1 et ω_2 .

Exemple A 1. : Soit la caractéristique X dont les densités conditionnelles pour ω_1 et ω_2 sont données en figure A.1.

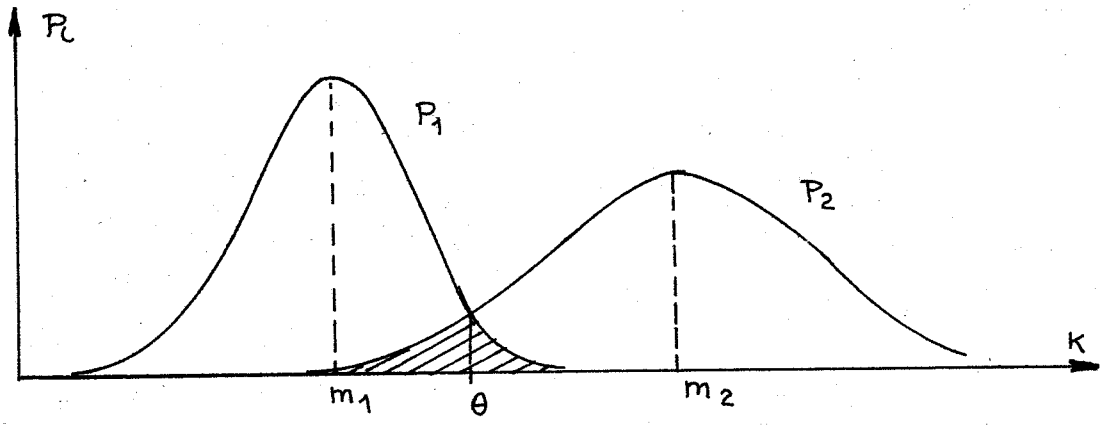


FIGURE A 1.

On veut partitionner K par un seuil unique θ de façon à avoir les deux règles d'identification :

"Si $X(e) < \theta$, alors $e \in \omega_1$ "

"Si $X(e) > \theta$, alors $e \in \omega_2$ "

La probabilité d'erreur sera alors :

$$Pr\ [\text{erreur}] = Pr\ (\omega_1) \int_{\theta}^{+\infty} P_1\ (t)\ dt \quad + \quad Pr\ (\omega_2) \int_{-\infty}^{\theta} P_2\ (t)\ dt\ .$$

On peut simplifier l'exemple en admettant des densités Gaussiennes de même écart type, et des classes à priori équiprobables. D'où :

$$Pr\ [\text{erreur}] = 1/2\ \left(1 - \Phi\ \left(\frac{\theta - m_1}{T}\right) - \Phi\ \left(\frac{\theta - m_2}{T}\right)\right) \quad ;$$

$$\text{avec } \Phi\ (x) = \int_{o}^{\infty} \frac{1}{\sqrt{2\pi}}\ e^{-t^2/2}\ dt$$

Même en positionnant θ à la valeur conduisant au minimum de cette expression (i.e., $\theta = (m_2 - m_1)/2$), l'erreur pourrait être très importante, par exemple 7 % pour $m_2 - m_1 = 3\sigma$.

Par ailleurs, l'introduction de plus de deux classes d'objets avec les règles du type :

"Si $X(e) < \theta$, alors e appartient au sous-ensemble de classes dont la moyenne est inférieure à θ ",

et la prise en compte de plusieurs caractéristiques ne modifie pas le résultat précédent : se donner comme seul degré de liberté, la position des seuils ne permettra qu'exceptionnellement de satisfaire une contrainte sur la probabilité erreur.

Segmentation par bi-partitions

On évitera cet inconvénient si l'on accepte une partition de l'espace de représentation avec des zones de non-classification, lesquelles correspondront aux sous-domaines où la probabilité d'erreur est plus grande que la contrainte.

Une procédure simple serait de définir une bi-partition de l'axe X relativement à chaque classe ω_i en :

(i) KS_i : zone où ω_i est "sélectionnée" par la caractéristique X ;

(ii) $K - KS_i$: zone où ω_i est "exclue" par X.

La partition globale de l'axe K sera le produit de ces bi-partitions :

$\bigtimes_i (KS_i ; K - KS_i) = (k_1, k_2, \ldots, k_r)$; avec les règles suivantes :

"Si $X(e) \in k_j$, alors $e \in \{\omega_i \mid k_j \subset KS_i\}$

Exemple A.2. : On obtient, pour l'exemple précédent, la partition suivante :

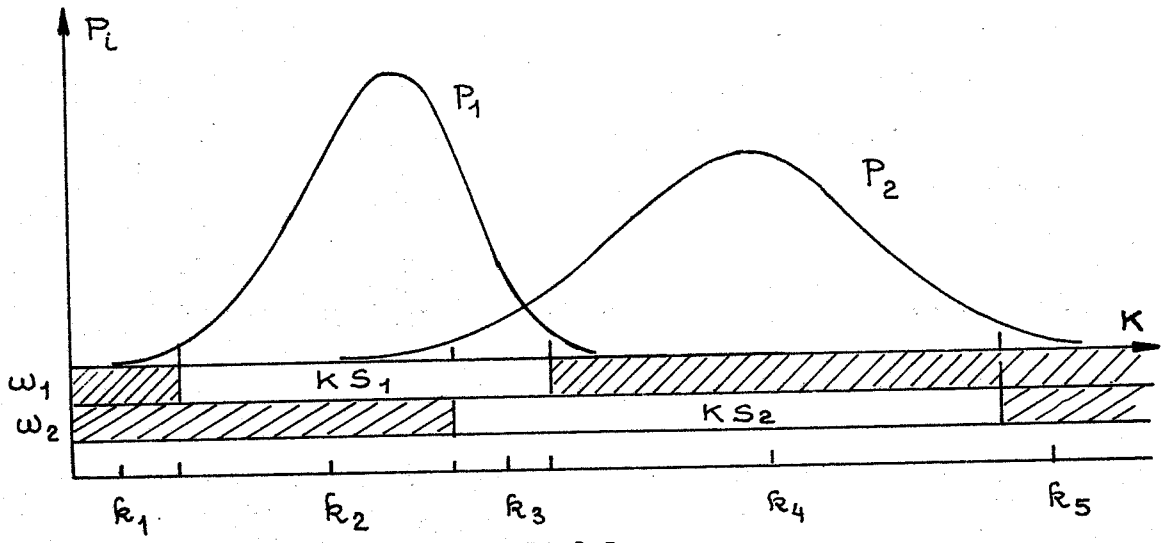


FIGURE A.2.

La composition de plusieurs caractéristiques, partitionnées comme précédemment, se fera suivant la règle de conjonction : il y aura identification de e à ω_i si l'intersection des sous-ensembles de classes sélectionnées par les caractéristiques évaluées est exactement ω_i. Si cette intersection est vide, ou bien si elle comporte plus d'une classe et qu'aucune caractéristique non encore évaluée ne reste disponible, le système concluera par la non-classification.

Dans l'exemple précédent, et si on ne dispose que d'une seule caractéristique, il y aura non-classification en k_1, k_3 et k_5.

Evaluons, à présent, la probabilité d'erreur d'un tel classifieur :

$Pr[\text{erreur}] = \sum_i Pr(\omega_i) \cdot Pr[\text{erreur}/\omega_i]$; et

$Pr[\text{erreur}/\omega_i] = \sum_{j \neq i} Pr[\text{identifier } \omega_j / \omega_i]$.

Dans le pire cas, il y aura identification de ω_j sachant ω_i si au moins une de r caractéristiques évaluées (par exemple X_1) exclut ω_i, et les (r-1) autres sélectionnent ω_i :

$$\Pr[\text{identifier } \omega_j/\omega_i] \leq \Pr[X_1(e) \text{ exclut } \omega_i/\omega_i] \times \Pr[X_2(e) \text{ sélectionne } \omega_i/\omega_i] \ldots \times \Pr[X_r(e) \text{ sélectionne } \omega_i/\omega_i] .$$

Si l'on définit les zones de section et d'exclusion, KS_i et KE_i, de chaque classe ω_i par :

$$\int_{KE_i} p_i(t).dt = \lambda \quad ; \quad \forall x, y : x \in KE, y \notin KE \Rightarrow p(x) < p(y) \; ; \text{ et } KS = K - KE$$

on aura :

$$\Pr[\text{identifier } \omega_j/\omega_i] \leq \lambda(1-\lambda)^{r-1} \quad ; \text{ pour r caractéristiques évaluées ;}$$

$$\Pr[\text{erreur}] \leq (m-1)\,\lambda\,(1-\lambda)^{r-1} \quad \text{pour m classes.}$$

Finalement, si η est la contrainte fixée sur la probabilité d'erreur, il suffit de prendre : $\lambda \leq \frac{\eta}{m-1}$ pour la respecter.

La non-classification correspond à deux cas de figure :

- l'intersection des sous-ensembles de classes sélectionnées est vide : la probabilité de cette situation admet le même majorant que la probabilité d'erreur ;

- deux (ou plus de deux) classes demeurent sélectionnées après l'évaluation de toutes les caractéristiques connues : la probabilité de cette situation dépend du cas particulier traité et ne peut être à priori majorée (pour une seule caractéristique et deux classes dont les densités se recouvrent largement, cette probabilité peut être arbitrairement proche de 1). On peut, cependant, la calculer assez simplement, et sa connaissance fournit à l'utilisateur une bonne indication du pouvoir séparateur du classifieur. Dans l'exemple précédent, elle correspond à :

$$\int_{k_3} [\Pr(\omega_1)\, p_1(t) + \Pr(\omega_2)\, p_2(t)]\, dt .$$

Ce classifieur est très facile à mettre en oeuvre, mais il a un inconvénient majeur : il conduit à une identification qui ne correspond pas toujours au maximum de vraisemblance. En effet, dans un ensemble de situations qui peuvent être de probabilité non négligeable, la classe identifiée n'est pas la plus probable, compte-tenu de l'information dont on dispose. Autrement dit, si au lieu de ne retenir que le segment auquel appartient la mesure effectuée, on conservait la valeur de cette mesure, un calcul des probabilités a postériori conduirait, avec les mêmes hypothèses, à une classification différente de celle obtenue par les règles de décision précédentes.

Exemple A.3. : Déterminons quelle peut être, dans un cas défavorable, la probabilité de ces situations. Reprenons, pour cela, l'exemple de deux classes ω_1 et ω_2 à priori équiprobables, avec deux caractéristiques X et X' de densités Gaussiennes normales (figure A.3.). Si s est tel que $\int_{m-s}^{m+s} p(t).dt = 1 - \lambda$, le domaine de sélection dans chaque cas est alors le segment $[\, m - s,\ m + s]$. Un exemple défavorable de recouvrement des distributions est celui de la figure A.4. avec :
$m_2 = m_1 + s$ et $m'_2 = m'_1 + 2\ s$.

Dans un tel système, il y aura identification à ω_1, alors que ω_2 est à postériori la plus probable si les trois conditions suivantes sont vérifiées :

. $X(e) = x$ tel que $x \in KS_1 \cap KS_2$; i.e. : $m \leqslant x \leqslant m + s$

. $X'(e) = x'$ tel que $x' \in KS'_1$ et $x' \notin KS'_2$; i.e. : $m'-s \leqslant x' \leqslant m'+s$

. $p_1(x)\ p'_1(x') \leqslant p_2(x)\ \ p'_2(x')$

Cette dernière relation conduit à :

$$P'_2(x') / P'_1(x') \ - \ P_1(x) \ / \ P_2(x) \geqslant 0$$

$$e^{2s(n'-m'-s)} \ - \ e^{-s(x-m-s/2)} \ \geqslant 0$$

$$2(x'-m') + x-m \ - \ 5s/2 \geqslant 0$$

La probabilité de cette situation est donc celle de trouver le point (x,x') dans le triangle hachuré (figure A.4.).

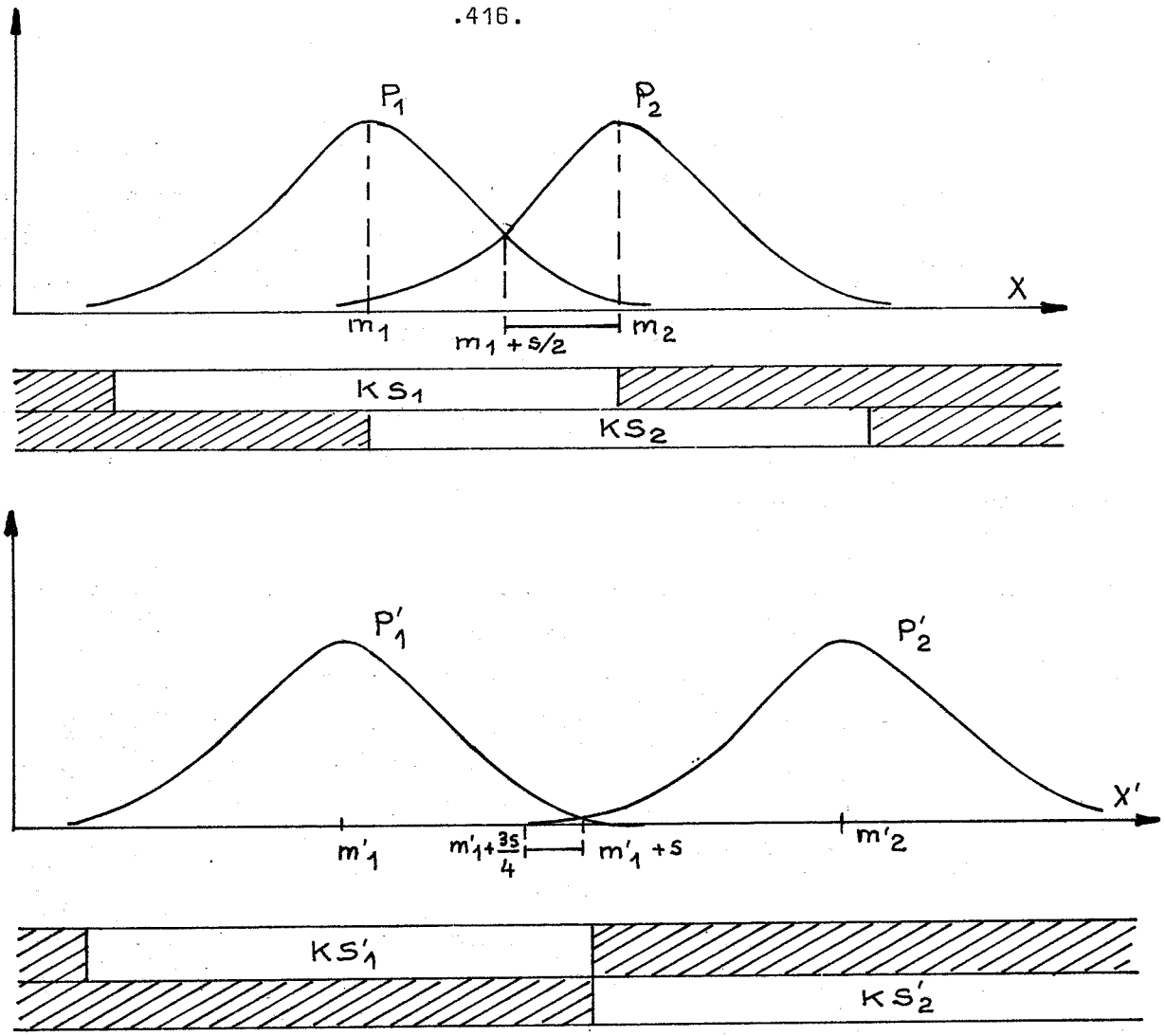


FIGURE A.3.

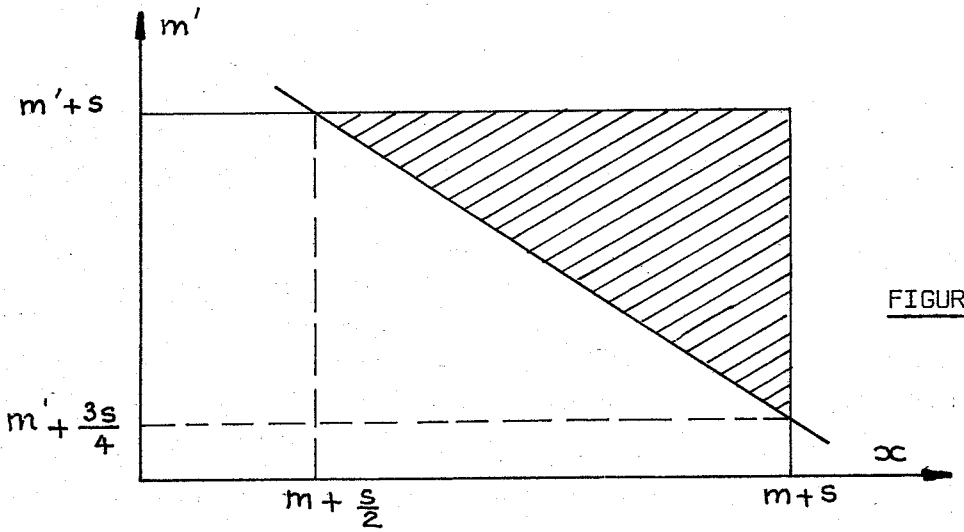


FIGURE A.4.

En tenant compte de la situation symétrique (identification à ω_2 alors que ω_1 est la plus probable), et pour s = σ (écart type), la probabilité de classification autre qu'au maximum de vraisemblance est alors :

$$\left[1/2 \int_{m+s+2}^{m+s} (p_1(t) + p_2(t)dt)\right] x \left[1/2 \int_{m'+3s/4}^{m'+s} (p'_1(t) + p'_2(t))dt\right] =$$

$$1/4 \times \Phi(1) \times (\Phi(5/4) - \Phi(3/4)) = 0.0103.$$

Ce contre-exemple établit la proposition 5.2. De plus, il est assez simple de montrer, qu'au pire des cas, la probabilité précédente croît avec le nombre de caractéristiques et de classes d'objets. Dans un exemple similaire de trois classes et trois caractéristiques, l'identification ne correspondra pas à la classe la plus probable dans 1.7 % des cas.

Pour résumer l'argumentation justifiant la méthode adoptée au chapitre 5, on cherche à retenir, autant que possible, l'avantage de la classification Bayesienne (minimisation de la probabilité d'erreur), tout en évitant ses inconvénients (lourdeur, complexité, difficulté d'apprentissage). Autrement dit, on voudrait une partition de l'espace de représentation qui permette une RFS par simple localisation d'une mesure relativement à des bornes (simplifiant ainsi l'apprentissage et la génération du classifieur), mais on ne s'autorise pas à négliger une information <u>disponible</u> quand cela détériore sensiblement les performances du système.

Il a été possible de satisfaire cette double exigence dans le cas de notre application, grâce à la marge de non-classification, importante relativement à la marge d'erreur que l'on s'autorise.

°0°

BIBLIOGRAPHIE

1. AGIN G.J., and DUDA R.O., "SRI vision research for advanced industrial automation", Proc. 2nd USA-JAPAN Computer Conference, pp.113-117, 1975.

2. AGRAWALA A.K., and KULKARNI A.V., "A sequential approach to the extraction of shape features", Computer Graphics and Image Processing, 6, pp.538-557, 1977.

3. AHO A.V., and CORASICK M.J., "Efficient string matching : an aid to bibliographic search", Comm. ACM, 18, 6, pp.333-340, June 1975.

4. AHO A.V., HIRSCHBERG D.S., and ULLMAN J.D., "Bounds on the complexity of the longest common subsequence problem", J. ACM, 23, 1, pp.1-12, Jan. 1976.

5. AHO A.V., and ULLMAN J.D., "The theory of parsing, translation and compiling", Vol.II : Compiling, Prentice-Hall, 1972.

6. ANDERSON R.H., "The use of production systems in RITA to construct personal computer "agents", SIGART Newsletter, 63, pp.23-28, 1977.

7. BAKER T.P., "A technique for extending rapid exact-match string matching to arrays of more than one dimension", SIAM, J. Computing, 8,4, pp.533-541, Nov. 1978

8. BANERJI R.B., "Some linguistic and statistical problems in pattern recognition", Pattern Recognition, 3, pp.409-419, 1971.

9 BARON M., "Documentation et homologation des programmes à l'aide des tables de décision", Thèse de Docteur-Ingénieur, U.S.M., Grenoble, 1973.

10. BATCHELOR B.G., (ed.), "Pattern recognition, ideas and practice", Plenum Press, 1978.

11. BAXTER L.D., "The complexity of unification", Ph.D. Thesis, Dept. of Computer Science, Univ. of Waterloo, Ontario, 1976.

12. Bell D.A., "Decision trees, tables and lattices", In Pattern Recognition, Ideas and Practice, Batchelor ed., Plenum Press, New-York, pp.119-141, 1978.

13. BELLMAN R., and DREYFUS S., "Applied dynamic programming", Princeton U. Press, 1962.

14. BERLINGER H., "The B^* tree search algorithm : a best-first proof procedure", Artificial Intelligence, 12, 1, pp.23-40, May 1979.

15. BERTELE U., and BRIOSCHI F., "Non-serial dynamic programming", Academic Press, 1972.

16. BOLLES R.C., "Locating partially visible objects : the local-feature-focus method", Proc. 1st Annual National Conf. on Artificial Intelligence, pp.41-43, Stanford, Aug. 1980.

17. BOURDEAU C., BRIOT M., PONS J.M., et TALOU J.C., "Etude et réalisation d'un extracteur rapide de contours fumés d'une image binarisée", Actes 3ème Congrès de Reconnaissance des Formes et d'Intelligence Artificielle de l'A.F.C.E.T., pp.539-551, Nancy, Sept. 1981.

18. BOYER R.S., and MOORE J.S., "A fast string searching algorithm", Comm. ACM, 20, 10, pp.762-772, Oct. 1977.

19. CANTRELL H.N., KING J., and KING F.E.H., "Logic structure tables", Comm. ACM, 4, pp.272-275, June 1961.

20. CAVOURAS J.C., "On the conversion of programs to decision tables : methods and objectives", Comm. ACM, 17, 8, pp.456-462, Aug. 1974.

21. CERNY E., MANGE D., and SANCHEZ E., "Synthesis of minimal binary decision trees", IEEE Trans. on Comput., C-28, 7, pp.472-482, July 1979.

22. CHANG C.L., and LEE R.C., "Symbolic logic and mechanical theorem proving ", Academic Press, 1972.

23. CHENAIS D., et TERRENOIRE M., "Pseudoquestionnaires", RIRO, R2, pp.43-54, 1971.

24. CHIN R., BAUDET P., and ARGENTIERO P., "An automated approach to the design of decision tree classifiers", Proc. 5th IEEE Conf. on Pattern Recognition, pp.660-665, Dec. 1980.

26. COOK S.A., "The complexity of theorem-proving procedures", Proc. 3rd ACM Symp. on Theory of Computing, pp.151-158, New-York, 1971.

27. DATHE G., "Conversion of decision tables by rule mask method without rule mask", Comm. ACM, 15, 10, pp.906-909, Oct. 1979.

28. DATTATREYA G.R., and SARMA V.V.S., "Decision tree design for pattern recognition including feature measurement cost", Proc. 5th IEEE Conf. on Pattern Recognition, pp.1212-1214, Dec. 1980.

29. DAVIS R., and KING J. "An overview of production systems", In Machine Intelligence, 8, Elcock and Michie, ed., pp.300-332, Wiley, 1976.

30. DENARDO E.V., "Sequential decision processes", Ph.D. Thesis, Northwestern Univ., Evanston, Illinois, 1965.

31. DESSIMOZ J.D., "Traitement des contours en reconnaissance de formes visuelles : application à la robotique", Thèse Ph.D., n°387, E.P.F., Lausanne, 1980.

32. DOUCET J.E., "Un programme de génération d'arbre de décision", Note Technique LAAS-SIS, 79.T.49, 1979.

33. DUBAY F., "Algorithmes de questionnaires réalisables optimaux au sens de différents critères", Thèse de 3ème Cycle, Lyon, 1967.

34. DUDA R.O., and HART P.E., "Pattern classification and scenes analysis", Wiley Interscience, 1973.

35. DUDA R.O., HART P.E., NILSSON N.J., REBOH R., SLOCUM J., and SUTHERLAND G., "Development of the PROSPECTOR consultation system for mineral exploration", Tech. Note, Artificial Intellience Center, SRI, Sept. 1978.

36. DUFRESNE P., "Elaboration d'un générateur de plans en vue d'une étude de l'apprentissage", Mémoire de D.E.A., L.A.A.S., Oct. 1982.

37. DUNCAN G.T., "Heterogeneous questionnaire theory", SIAM J. Applied Math., 27, 1, pp.59-71, July 1974.

38. DUNCAN G.T., "Optimal diagnostic questionnaires", Operations Research, 23, 1, pp.22-32, Jan. 1975.

39. EGLER J.F., "A procedure for converting logic table conditions into an efficient sequence of test instructions", Comm. ACM 6, 8, pp.510-514, Sept. 1963.

40. ERNST G.W., and NEWELL A., "GPS : a case study in generality and problem solving", Academic Press, 1969.

41. FARRENY H., "Un système pour l'expression et la résolution de problèmes orientés vers le contrôle de robots", Thèse de Doctorat d'Etat, U.P.S., Toulouse, 1980.

42. FIKES R.E., HART P.E., and NILSSON N.J., "Learning and executing generalized robots plans", Techn. Note 70, Artificial Intelligence Center, SRI, July 1972.

43. FIKES R.E., and NILSSON N.J., "STRIPS : a new approach to the application of theorem proving to problem solving", Artificial Intelligence, 2, pp.189-208, 1971.

44. FISCHER M.J., and PATERSON M.S., "String matching and other products", SIAM, Proc. AMS, 7, pp.113-127, 1974.

45. FORGY C.L., and McDERMOTT J., "OPS : a domain independant production system language", Proc. 5th IJCAI, Cambridge, Mass., 1977.

46. FORGY C.L., "On the efficient implementation of productions systems", Ph.D. Thesis, Dep. of Computer Science, Carnegie-Mellon Univ., Pittsburgh, 1979.

47. FORGY C.L., "OPS4 user's manual", Tech. Report, Dep. of Computer Science, Carnegie-Mellon Univ., Pittsburgh, July 1979.

48. FU K.S., CHIEN Y.T., and CARDILLO G.P., "A dynamic programming approach to sequential pattern recognition", IEEE Trans. Elec. Comp., EC-16, 6, pp.790-803, Dec. 1967.

49. FU K.S., "Sequential methods in pattern recognition and machine learning", Academic Press, 1968.

50. FU K.S. (ed.), "Digital pattern recognition", Communication and Cybernetics, 10, 1976.

51. FU K.S., "Syntactic(linguistic) pattern recognition", In Digital Pattern Recognition, FU ed., Communication and Cybernetic, 10, pp.95-134, 1976.

52. FU K.S., "Recent developments in pattern recognition", IEEE Trans. Comp., C-29, 10, pp.845-855, Oct. 1980.

53. GALIL Z., "On improving the worst case running time of the Boyer-and-Moore string matching algorithm", Comm. ACM, 9, 22, pp.505-508, Sept. 1979.

54 GALLAIRE H., "Etude des systèmes de génération de plans d'actions pour des robots", Rapport ONERA-CERT n°1/3107/DERI, 1978.

55. GALLAIRE H., "Cours Prolog" (I.R.R., U.P.S., Toulouse), NATO A.S.I. on Automatic Program Construction, Reidel Ed., Bonas , Sept. 1981.

56. GANAPATHY S., and RAJARAMAN V., "Information theory applied to the conversion of decision tables to computer programs", Comm. ACM, 16, 9, pp.532-539, Sept. 1973.

57. GAREY M.R., "Simple binary identification problems", IEEE Trans. on Computers, C-21, 6, pp.588-590, June 1972.

58. GAREY M.R., "Optimal binary identification procedures", SIAM J. Applied Math., 23, 2, pp.173-186, Sept. 1972.

59. GAREY M.R., and GRAHAM R.L., "Performance bounds on the splitting algorithm for binary testing", Acta Informatica, 3, pp.347-355, 1974.

60. GAREY M.R., GRAHAM R.L., and JOHNSON D.S., "Performance guarantees for scheduling algorithms", Operations Research, 26, 1, pp.3-21, Jan. 1978.

61. GAREY M.R., and JOHNSON D.S., "The complexity of near optimal graph coloring", J. ACM, 23, 1, pp.43-49, Jan. 1976.

62. GAREY M.R., and JOHNSON D.S., "Strong NP-completeness results : motivation, examples and implications", J. ACM, 25, 3, pp.499-508, July 1978.

63. GAREY M.R., and JOHNSON D.S., "Computer and intractability - a guide to the theory of NP-completeness", Freeman, 1978.

64. GASHNIG J., "Performance measurement and analysis of certain search algorithms", Ph.D. Thesis, Dept. of Computer Science, Carnegie-Mellon Univ., Pittsburgh, May 1979.

65. GASHNIG J., "A problem similarity approach to divising heuristics : first results", Proc. 6th IJCAI, pp.301-307, Tokyo, Sept. 1979.

66. GELPERIN D., "On the optimality of A^*", Artificial Intelligence, 8, pp.69-76, 1977.

67. GHALLAB M., "Consistance, complétude et traduction optimale des tables de décision", RAIRO, Rech. Op., 12, 1, pp.61-84, Fev. 1978.

68. GHALLAB M., "Approximate solution for the conversion of decision tables problem", Memo, ERL, M-79, 60, Univ. of California, Berkeley, Sept. 1979.

69. GHALLAB M., "Near optimal decision trees for matching patterns in inference and planning systems", 2nd Int. Meeting on Artificial Intelligence, Leningrad, Oct. 1980.

70. GHALLAB M., "Decision trees for optimizing pattern-matching algorithms in production systems", Proc. 7th IJCAI, pp.310-312, Vancouver, 1981.

71. GHALLAB M., "Algorithmes quasi-admissibles de recherche heuristique dans les graphes et les hypergraphes", Note Interne, LAAS, 81.I.49, Nov. 1981.

72. GHALLAB M., and ALLARD D., "Near admissible heuristic search algorithms", Proc. 2nd World Conf. on Mathematics at the Service of Man, Las Palmes, June 1982.

73. GHALLAB M., and GIRALT G., "A decision method for object identification in robotics", Proc. 17th IEEE Conf. on Decision and Control, San Diego, Jan. 1979.

74. GIRALT G., "La robotique : possibilités et avenir", La Pensée, 224, pp.74-84, Nov. 1981.

75. GIRALT G., GHALLAB M., and STUCK F., "Object identification and sorting with an optimal sequential pattern recognition method", Proc. 9th ISIR, Washington DC, March 1979.

76. GIRALT G., GHALLAB M., STUCK F., et GIRAUD A., "Système auto-adaptatif de perception optoélectronique pour l'identification et la manipulation de pièces en robotiques", Actes 2ème Conf. A.F.C.E.T. en Reconnaissance de Formes et Intelligence Artificielle, pp.209-216, Toulouse, Sept. 1979.

77. GONDRAN M., et MINOUX M., "Graphes et algorithmes", Eyrolles, 1979.

78. GREEN C., "Application of theorem proving to problem solving", Proc. 1st IJCAI, Washington DC, May 1969.

79. GUIBAS L.J., and ODLYZKO A.M., "A new proof of the linearity of Boyer-Moore string searching algorithm", Proc. IEEE 18th Symp. on Foundations of Computer Sciences, pp.189-195, Oct. 1977.

80. HALL P.A., "Equivalence between and/or graphs and context-free grammars", Comm. ACM, 16, 7, pp.444-445, July 1973.

81. HARRIS L.R., "The heuristic search under conditions of error", Artificial Intelligence, 5, pp.217-234, 1974.

82. HART P.E., NILSSON N.J., and RAPHAEL B., "A formal basis for the heuristic determination of minimal cost paths", IEEE Trans. SSC, 4, pp.100-107, 1968.

83. HART P.E., NILSSON, N.J., and RAPHAEL B., "Correction to : a formal basis for the heuristic determination of minimal cost paths", SIGART Newsletter, 3, pp.28-29, 1972.

84. HARTMANN C.R.P., VARSHNEY P.K., MEHROTRA K.G., and GERBERICH C.L., "Application of information theory to the construction of efficient decision trees", IEEE Trans. on Information Theory, IT-28, 4, pp.565-574, July 1982.

85. HAYES P.J., "In defense of logic", Proc. 6th IJCAI, pp.559-565, Tokyo, 1977.

86. HAYES-ROTH F., and MOSTOW D.J., "An automatically compilable recognition network for structured patterns, "Proc. 4th IJCAI, Tbilissi, 1975.

87. HAYES-ROTH F., WATERMAN D.A., and LENAT D.B., "Principles of pattern directed inference systems", In Pattern-Directed Inference Systems, Waterman and Hayes-Roth ed., Academic Press, 1978.

88. HENRICHON E.G., and FU K.S., "A non-parametric partitioning procedure for pattern classification", IEEE Trans. on Comp., C-18, 7, pp.614-624, July 1969.

89. HENSCHEN L.J., and NAQVI S.A., "An improved filter for literal indexing in resolution systems", Proc. 6th IJCAI, pp.528-529, Vancouver, 1981.

90. HERZOG T.N., "Generating uniform pseudo-random numbers", APL Quote Quad, 12, 2, pp.22-23, Dec. 1981.

91. HIRCHBERG D.S., "Algorithms for the longest common subsequence problem", J. ACM, 24, 4, pp.664-675, Oct. 1977.

92. HOFFMAN C.M., and O'DONNELL M.J., "Pattern matching in trees", J. ACM, 29, 1, pp.68-95, Jan. 1982.

93. HOFSTADTER D.R., "Mathematical games", Scientific American, May 1981.

94. HOPCROFT J.E., and ULLMAN J.D., "Formal languages and their solution to automata", Addison-Wesley, 1969.

95. HUET G., "Résolution d'équation dans des langages d'ordre 1,2,... ω ", Thèse de Doctorat d'Etat, Université Paris VII, Sept. 1976.

96. HUGHES M.L., SHANK R.M., and STERN E.S., "Decision tables", MDI Publications, Pennsylvania, 1968.

97. HYAFIL L., and RIVEST R.L., "Constructing optimal binary decision trees is NP-complete", Information Processing Letters, 5, 1, pp.15-17, May 1976.

98. HUYN N., DECHTER R., and PEARL J., "Probabilistic analysis et the complexity of A^*", Artificial Intelligence, 15, 3, pp.241-254, Dec. 1980.

99. IBARAKI T., "Theoretical comparison of search strategies in branch-and-bound algorithms", Internat. J. of Computer and Information Sciences, 5, 4, pp.315-344, 1976.

100. IBARAKI T., "The power of dominance relations in branch-and-bound algorithms", J. ACM, 24, 2, pp.264-279, April 1977.

101. IBARAKI T., "Branch-and-bound procedure and state space representation of combinational optimization problems", Information and Control, 36, 1, pp.1-27, 1978.

102. IBARRA O.H., and KIM C.E., "Fast approximation algorithms for the knapsack and sum of subset problem", J. ACM, 22, 4, pp.463-468, Oct. 1975.

103. IBRAMSHA M., and RAJARAMAN V., "Detection of logical errors in decision table programs", Comm. ACM, 21, 12, pp.1016-25, Dec. 1978.

104. JARVIS J.M., "An analysis of programming via decision table compilers", SIGPLAN Notices, 6, 8, pp.20-32, Sept. 1971.

105. JOHNSON D.S., "A note on Dijkstra's shortest path algorithm", J. ACM, 20, 7, pp.385-388, July 1973.

106. JOHNSON D.S., "Approximation algorithms for combinational problems", J. of Computer and Systems Sciences, 9, pp.256-278, 1974.

107. KANAL L.N., "Patterns in pattern recognition 1968-1974", IEEE Trans. Information Theory, IT-20, 6, pp.697-722, Nov. 1974.

108. KANAL L.N., "On hierarchical classifier, theory and interactive design", In Applications of Statistics, Krishnaïsh ed., North Holland, 1977.

109. KANAL L.N., "Problem-solving models and search strategies for pattern recognition", IEEE Trans. PAMI, 1, 2, pp.193-201, April 1979.

110. KARP R.M., "Reducibility among combinational problems", Proc. Symp. Complexity of Computer Computation, Miller and Thatcher ed., pp.85-104, Plenum Press, 1972.

111. KARP R.M., "On the computational complexity of combinatorial problems", Networks, 5, pp.45-68, 1975.

112. KARP R.M., "The probabilistic analysis of some combinatorial search algorithms", In Algorithms and Complexity : New Directions and Recent Results, Traub. ed., pp.1-20, Academic Press, 1976.

113. KARP R.M. "Probabilistic analysis of a canonical numbering algorithm for graphs", Proc. Symp. in Pure Mathematics, 34, pp.365-378, 1979.

114. KARP R.M., "Linear expected-time algorithms for connectivity problems", J. of Algorithms, 1, pp.374-393, 1980.

115. KARP R.M., and HELD M., "Finite-state processes and dynamic programming", SIAM, J. Applied Math., 15, 3, pp.695-718, May 1967.

116. KING P.J.H., "Conversion of decision tables to computer programs by rule mask techniques", Comm. ACM, 9, 11, pp.796-801, Nov. 1966.

117. KING P.J.H., "Ambiguity in limited entry decision table", Comm. ACM, 11, 10, pp.680-684, Oct. 1968.

118. KING P.J.H., "The interpretation of limited entry decision table format and relationships among conditions", Computer J., 12, 4, pp.320-324, Nov. 1969.

119. KING P.J.H., and JOHNSON R.G., "Some comments on the use of ambiguous decision tables and their conversion to computer programs", Comm. ACM, 16, 5, pp.287-290, May 1973.

120. KLEENE S., "Mathematical logic", Wiley, 1967.

121. KNUTH D.E., "The art of computer programming", vol.1 : Fundamental Algorithms, Addison-Wesley, 1968.

122. KNUTH D.E., "The art of computer programming", vol.2 : Semi-numerical Algorithms, Addison-Wesley, 1969.

123. KNUTH D.E., "The art of computer programming", vol.3 : Sorting and Searching, Addison-Wesley, 1973.

124. KNUTH D.E., MORRIS J.H., and PRATT V.R., "Fast pattern matching in strings", SIAM J. on Computing, 6, 2, pp.323-350, June 1977.

125. KOHLER W.H., and STEIGLITZ K., "Characterization and theoretical comparison of branch and bound algorithms for permutation problems", J. ACM, 21, 1, pp.140-156, Jan. 1974.

126. KOHLER W.H., and STEIGLITZ K., "Enumerative and iterative computational approach", In Computer and Job-Shop Scheduling Theory, Coffman ed., pp.229-298, Wiley Interscience, 1975.

127. KONOLIDGE K., "An inference net compiler for the prospector rule-based consultation system", Proc. 6th IJCAI, pp.487-489, Tokyo, 1979.

128. KORTE B., "Approximate algorithms for discrete optimization problems", Annals of Discrete Mathematics, 4, pp.85-120, 1979.

129. KOWALSKI R., "Predicate logic as a programming language", Proc. 3rd IFIP Congress, Stockholm, 1974.

130. KOWALSKI R.A., "Logic for problem solving", Elsevier, 1979.

131. KULKARNI A.V., "Optimal and heuristic synthesis of hierarchical classifiers", Ph.D. Thesis, Univ. of Maryland, 1976.

132. LARSON L.E., "Use of decision tables in multiprocessing environments", Ph.D. Thesis, State Univ. of New-York, Binghanton, 1979.

133. LASSERRE C., "Apport de la logique mathématique dans les systèmes de décision en robotique", Thèse de Docteur-Ingénieur, U.P.S., Toulouse, 1978.

134. LAURIERE J.L., "Représentation et utilisation des connaissances : les systèmes experts", Techniques et Sciences Informatiques, 1, 1, pp.25-42, Jan. 1982.

135. LAURIERE J.L., "Représentation et utilisation des connaissances : représentation des connaissances", Technique et Science Informatiques, 1, 2, pp.109-133, Mars 1982.

136. LAWLER E.L., "Fast approximation algorithms for knapsack problems", Proc. 18th Ann. Symp. on Foundations of Computer Science, pp.206-213, Long Beach, 1977

137. LAWLER E.L., and WOOD D.E., "Branch-and-bound methods : a survey", Operation Research, 14, 4, pp.699-719, July 1966.

138. LEMAITRE M., "Réalisations optimales et heuristiques de questionnaires", Thèse de Docteur-Ingénieur, U.P.S., Toulouse, 1975.

139. LENSTRA J.K., and RINNOOY-KAN A.H.G., "On the expected performance of branch-and-bound algorithms", Operations Research, 26, 2, pp.347-349, March 1979.

140. LEW A., "Optimal conversion of extended entry decision tables with general cost criteria", Comm. ACM, 21, 4, pp.269-279, April 1978.

141. LIEBERMAN H., "A preview of ACT 1", MIT-Artificial Intelligence Lab., Working Paper, 1980.

142. LONDON K., "Decision tables : a practical approach for data processing", Auerbach Publishers, 1972.

143. LOVELAND D.W., "Selecting optimal test procedures from incomplete test sets", Proc. 1st Int. Symp. on Policy Analysis and Information Systems, pp.228-235, Durham, 1979.

144. MAES R., "An algorithmic approach to the conversion of decision grid charts into computed decision tables", Comm. ACM, 23, 5, pp.280-293, May 1980.

145. MARTELLI A., "An application of heuristic search methods to edge and contour detection", Comm. ACM, 19, 2, pp.73-83, Feb. 1976.

146. MARTELLI A., "On the complexity of admissible search algorithms", Artificial Intelligence, 8, pp.1-13, 1977.

147. MARTELLI A., and MONTANARI U., "Additive and/or graphs", Proc. 3rd IJCAI, Stanford, pp.1-11, 1973.

148. MARTELLI A., and MONTANNARI U., "From dynamic programming to search algorithms with functional costs", Proc. 4th IJCAI, pp.345-350, Tbilissi, Sept. 1975.

149. MARTELLI A., and MONTANARI U., "Optimizing decision trees through heuristically guided search", Comm. ACM, 21, 12, pp.1025-39, Dec. 1978.

150. MARTELLI A., and MONTANARI U., "An efficient unification algorithm", ACM, TOPLAS, 4, 2, pp.258-282, April 1982.

151. McCLUSKEY G.J., "Introduction to the theory of switching circuits", Mc Graw-Hill, 1965.

152. McDANIEL H., "Decision table software", Brandon-Systems Press, Princeton, 1970.

153. McDANIEL H., "Applications of decision tables", Brandon-Systems Press, 1970.

154. McDERMOTT J., "Review of Nilsson's book : principles of artificial intelligence", Artificial Intelligence, 15, 1, p.127, Nov. 1980.

155. McDERMOTT J., NEWELL A., and MOORE J., "The efficiency of certain production systems implementation", In Pattern-Directed Inference Systems, Waterman and Hayes Roth ed., Academic Press, 1978.

156. McKEE J.W., and AGGARWAL J.K., "Computer recognition of partial views of curved objects", IEEE Trans. Computers, C-26, 8, pp.790-800, 1977.

157. MEISEL W.S., and MICHALOPOULOS D.A., "A partitioning algorithm with application in pattern classification and the optimization of decision trees", IEEE Trans. Comp., C-22, 1, pp.93-103, Jan. 1973.

158. MENON M.J., and HSIAO D.K., "Design and analysis of a relational join operation for VLSI", Proc. 7th Int. Conf. on Very Large Data Bases, Cannes, Sept. 1981.

159. METZNER J.R., and BARNES B.H., "Decision table languages and systems", Academic Press, 1977.

160. MITCHELL T.M., UTGOFF P.E., NUDEL B., and BANERJI R., "Learning problem-solving heuristics through practice", Proc. 7th IJCAI, Vancouver, April 1981.

161. MITTEN L.G., "Branch-and-Bound methods : general formulation and properties", Operations Research, 18, 1, pp.24-34, 1970.

162. MONTALBANO M., "Tables, flowcharts and program logic", IBM Systems J., pp.51-63, Sept. 1962.

163. MONTANARI H., "Heuristically guided search and chromosome matching", Artificial Intelligence, 4, pp.227-245, 1970.

164. MORET B.M.E., THOMASON M.G., and GONZHEZ R.L., "The activity of a variable and its relation to decision trees", ACM, TOPLAS, 2, 4, pp.580-595, October 1980.

165. MUTHUKRISHNAN C.R., and RAJARAMAN V., "On the conversion of decision trees to computer programs", Comm. ACM, 13, 6, pp.347-351, June 1970.

166. MYERS H.R., "Compiling optimized code from decision tables", IBM J. Res. Develop., 16, 5, pp.489-503, Sept. 1972.

167. NEWELL A., and McDERMOTT J., "PSG manual", Techn. Report, Dept. of Computer Science, Carnegie-Mellon Univ., Pittsburgh, 1975.

168. NEWELL A., and SIMON H.A., "Human problem solving", Prentice-Hall, 1972.

169. NII H.P., and FEIGENBAUM E.A., "Rule-base understanding of signals", In Pattern-Directed Inference Systems, Waterman and Hayes-Roth ed., pp.483-502, Academic Press, 1978.

170. NILSSON N.J., "Problem-solving methods in artificial intelligence", Mc Graw-Hill, 1971.

171. NILSSON N.J., "Principles of artificial intelligence", Tioga, P.A., 1979.

172. PANKHURST R.J., "A computer program for generating diagnostic keys", Computer J., 13, pp.145-151, 1970.

173. PAVLIDIS T., "Structural pattern recognition", Springer-Verlag, 1977.

174. PAVLIDIS T., and ALI F., "A hierarchical syntactic shape analyser", IEEE Trans. PAMI, 1, 1, pp.2-9, Jan. 1979.

175. PAVLIDIS T., "The use of a synthetic shape analyser for contour matching", IEEE Trans. PAMI, 1, 3, pp.307-310, March 1979.

176. PAYNE H.J., and MEISEL W.S., "An algorithm for constructing optimal binary decision trees", Trans. on Computers, C-26, 9, pp.905-916, Sept. 1977.

177. PEARL J., "Studies in heuristics. Part 1 : Three variations on a theme in A*", UCLA, ENG-CSL-7934, Los Angeles, June 1979.

178. PEARL J., "Capacity and error estimates for boolean classifiers with limited complexity", IEEE Trans. PAMI, 1, 4, pp.350-355, Oct. 1979.

179. PEARL J., "The utility of precision in search heuristics", Tech. Report, UCLA, ENG-CSL-8065, Los Angeles, Dec. 1980.

180. PEARL J., "Heuristic search theory : survey of recent results", Proc. 7th IJCAI, Vancouver, Aug. 1981.

181. PEARL J., and KIM J., "Studies in heuristics. Part 2 : Experiments on semi-admissible heuristics", UCLA, ENG-CS1-7961, Los Angeles, Sept. 1979.

182. PEARL J., and KIM J.H., "Studies in semi-admissible heuristics", IEEE Trans. PAMI, 4,4, pp.392-400, July 1982.

183. PEDNAULT E.P.D., ZUCKER S.W., and MERESAN L.V., "On the independence assumption underlying subjective bayesian updating", Artificial Intelligence, 16, pp.213-222, 1981.

184. PICARD C.F., "Theorie des questionnaires", Gauthier-Villars, 1965.

185. PICARD C.F., "Graphes et questionnaires", Gauthier-Villars, 1972.

186. PICARD C.F., "Graphs and questionnaires", North-Holland, 1980.

187. POHL I., "First results on the effect of error in heuristic search", In Machine Intelligence 5,Meltzer and Michie ed., pp.219-236, Edinburgh Univ. Press, 1970.

188. POHL I., "Practical and theoretical considerations in heuristic search algorithms", In Machine Intelligence 8, Elcock and Michie ed., Horwood, pp.55-71, 1977.

189. POLLACK S.L., "Conversion of limited entry decision tables to computer programs", Comm. ACM, 8, 11, pp.677-682, Nov. 1965.

190. POLLACK S.L., HICKS H., and HARRISON W.J., "Decision tables : theory and practice", Wiley, 1971.

191. POMIAN C., "Contribution à la definition et à l'implémentation d'un interprète du langage PLASMA", Thèse de Doctorat de Spécialité, U.P.S., Toulouse, Mars 1980.

192. POOCH U.W., "Translation of decision tables", Computing Surveys, 6, 2, pp.125-151, June 1974.

193. POSA L., "Hamilton circuits in random graphs", Discrete Mathematics, 14, 4, pp.359-364, 1976.

194. PRAJOUX R., "L'apport de l'intelligence artificielle en robotique", Annales des Mines, pp.63-70, Mai 1982.

195. PRAJOUX R., FARRENY H., et GHALLAB M., "Les robots : fonction, décision et intelligence artificielle", Hermes Publishing (à paraître 1983).

196. PRESS L.J., "Conversion of decision tables to computer programs", Comm ACM, 8, 6, pp.385-390, June 1965.

197. RABIN J., "Conversion of limited entry decision tables into optimal decision trees : fundamental concepts", Sigplan Notice, 6, 8, pp.68-74, Sept. 1971.

198. RABIN M.O., "Complexity of computations", Comm. ACM, 20, 9, pp.625-633, Sept. 1977.

199. RAIFA H., "Decision analysis : introductory lectures in choices under uncertainty", Addison-Wesley, 1968.

200. RAULEFS P., SIEKMAN J., SZABO P., and UNVERICHT E., "A short survey on the state of the art in matching and unification problems", AISB Newsletter, pp.17-21, Dec. 1978.

201. REINWALD L.T., and SOLAND R.M., "Conversion of limited-entry decision tables to optimal computer programs. I : Minimum average processing time", J. ACM, 13, 3, pp.339-358, July 1966.

202. REINWALD L.T., and SOLAND R.M., "Conversion of limited entry decision tables to optimal computer programs. II : Minimum storage requirement", J. ACM, 14, 4, pp.742-755, Oct. 1967.

203. RIVEST R.L., "On the worst case behavior of string-searching algorithms", SIAM J. Comput., 6, 4, pp.669-674, Dec. 1977.

204. ROBINSON J.A., "A machine oriented logic based on the resolution principle", J. ACM, 12, pp.23-41, Jan. 1965.

205. ROBINSON J.A., "Computational logic : the unification computation", In Machine-Intelligence 6, Meltzer and Michie, ed., Edinburgh U. Press, 1971.

206. ROCHE C., "Information utile en reconnaissance des formes et en compression de données. Application à la génération automatique de systèmes de reconnaissance optique et acoustique", Thèse de Docteur-d'Etat, Université Paris VI, 1972.

207. ROSENSCHEIN S.J., "The production system : architecture and abstraction", In Pattern-Directed Inference Systems, Waterman and Hayes-Roth ed., Academic Press, 1978.

208. ROUSSEL P., "Prolog, manuel de référence et d'utilisation", Rapport, Groupe d'Intelligence Artificielle, Université d'Aix-Marseille, 1975.

209. ROY B., "Algébrie moderne et théorie des graphes", DUNOD, 1970.

210. RULIFSON J.F., DERKSEN J.A., and WALDINGER R.J., "QA4 : a procedural calculus for intuitive reasoning", Techn. note 73, Artificial Intelligence Center, SRI, 1972.

211. SACERDOTI E.D., "Planning in a hierarchy of abstraction space", Proc. 3rd IJCAI, Stanford, Aug. 1973.

212. SACERDOTI E.D., "A structure for plans and behavior", American Elsevier, 1977.

213. SACERDOTI E.D., FIKES R.E., REBOH R., SAGALOWICZ D., WALDINGER R.J., and WILBER B.M., "QLISP, A language for interactive development of complex systems", Proc. Nat. Comput. Conf., pp.344-356, AFIPS Press, 1976.

214. SAHNI S., "Computationally related problems", SIAM J. Comput., 3, 4, pp.262-279, Dec. 1974.

215. SAHNI S., "Approximate algorithms for the 0/1 knapsack problem", J. ACM, 22, 1, pp.115-124, Jan. 1975.

216. SAHNI S., and GONZALEZ T., "P-complete approximation problems", J. ACM, 23, 3, pp.555-565, July 1976.

217. SAHNI S., "General technics for combinatorial approximation", Operations Research, 25, 6, pp.920-936, Nov. 1977.

218. SCHUMACHER H., and SEVCIK K.C., "The synthetic approach to decision table conversion", Comm. ACM, 19, 6, pp.343-351, June 1976.

219. SCHWARTZ B.M., "LISP 1.5. decision tables implemented for a serial computer and proposed for parallel computers", SIGPLAN Notices, 6, 8, pp.93-103, Sept. 1971.

220. SETHI I.K., and CHATTERJEE B., "Efficient decision tree design for discrete variable pattern recognition problems", Pattern Recognition, 9, pp.197-206, 1977.

221. SETHI I.K., and CHATTERJEE B., "Conversion of decision tables to efficient sequentiel testing procedures", Comm. ACM, 23, 5, pp.579-585, May 1980.

222. SHWAYDER K., "Conversion of limited entry decision tables to computer programs ; a proposed modification to Pollack's algorithm ", Comm. ACM, 14, 2, pp.69-73, Feb. 1971.

223. SHWAYDER K., "Extending the information theory approach to converting limited entry decision tables to computer programs", Comm. ACM, 17, 9, pp.532-537, Sept. 1974.

224 SIMON H.A., and KADANE J.B., "Problems of computational complexity in artificial intelligence", Proc. Symp. New Directions and Recent Results in Algorithms and Complexity, Carnegie-Mellon Univ., TRAUB éd., April 1976.

225. SIMON J.C., and ROCHE C., "Application of questionnaire theory to pattern recognition", 2nd IJCAI, pp.391-401, London, Sept. 1971.

226. SLAGLE J.R., and LEE R.C.T., "Application of game tree searching techniques to sequential pattern recognition", Comm. ACM, 14, 2, pp.103-110, Feb. 1971.

227. SOBEK R.P., "Compiling patterns for a production system : implementation considerations", Internal Note LAAS (à paraître).

228. STEFIK M., AIKINS J., BALZER R., BENOIT J., BIRBAUM L., HAYES-ROTH F., SACERDOTI E.D., "The organization of expert systems, a tutorial", Artificial Intelligence, 8, pp.135-173, 1982.

229. STOFFEL J.C., "A classifier design technique for discrete variable pattern recognition problems", IEEE Trans. Comp., C-23, 4, pp.428-444, April 1974.

230. STRUNZ H., "The development of decision trees via parsing of complex decision situations", Comm. ACM, 16, 6, pp.366-369, June 1973.

231. STUCK F., "Réalisation d'un système adaptatif de traitement d'images pour l'identification et la localisation de pièces en robotique", Thèse de 3ème Cycle, U.P.S., Toulouse, 1980.

232. TALOU J.C., Thèse de 3ème Cycle, U.P.S., Toulouse (à paraître).

233. TERRENOIRE M., "Un modèle mathématique de processus d'interrogation : les pseudo-questionnaires", Thèse de Doctorat d'Etat, U.S.M., Grenoble, 1970.

234. TRIGOBOFF M.L., "IRIS : a framework for the construction of clinical consultation systems", Ph.D. Thesis, Rutgers Univ., New Brunswick, 1978.

235. TSAI W.H., and FU K.S., "Attributed grammar - a tool for combining syntactic and statistical approaches to pattern recognition", IEEE Trans. SMC, 10, 12, pp.837-885, Dec. 1980.

236. ULLMAN J.D., "Principles of database systems", Computer Science Press, 1980.

237. VANDERBRUG G.J., "Problem representations and formal properties of heuristic search", Information Sciences, 11, pp.279-307, 1976.

238. VEINOTT C.G., "Programming decision tables in Fortran, Cobol or Algol", Comm. ACM, 9, 1, pp.31-35, Jan. 1966.

239. VERHELST M., "The conversion of limited entry decision tables to optimal and near optimal flowcharts : two new algorithms", Comm. ACM, 15, 11, pp.974-980, Nov. 1972.

240. WARREN D.H.D., "WARPLAN : a system for generating plans", Dept. of Computational Logic, Memo 76, Univ. of Edinburgh, 1974.

241. WARREN D.H.D., PEREIRA L.M., and PEREIRA F., "PROLOG : the language and its implementation compared with LISP", ACM Symp. on Artificial Intelligence and Programming Languages, SIGPLAN/SIGART, 64, pp.109-115, Aug. 1977.

242. WATERMAN D.A., and HAYES-ROTH F., "An overview of pattern-directed inference systems", In Pattern-directed Inference Systems, Waterman and Hayes-Roth ed., Academic Press, 1978.

243. YAO A.C., "The complexity of pattern matching for a random string", SIAM J. Comput., 8, 3, pp.368-387, Aug. 1979.

244. YASUI T., "Conversion of decision tables into decision trees", Ph.D. Thesis, Rep. 501, Univ. of Illinois, Feb. 1972.

245. ZUCKER S.W., "Production systems with feedback", In Pattern-Directed Inference Systems, Waterman and Hayes-Roth ed. Academic Press, 1978.

246. "La décision : agrégation dynamique des ordres de préférences", Colloque Int., Aix-en-Provence, Editions du C.N.R.S., Juillet 1967.