Contribution à la visualisation des connaissances par des graphes dans une mémoire d'entreprise : application sur le serveur Atanor

Bruno Pinaud

Ecole Centrale de Nantes Université de Nantes Ecole des Mines de Nantes

Ecole Doctorale STIM
Sciences et Technologies de l'Information et des Matériaux
Année 2006
N° E.D. : 366–256

THESE DE DOCTORAT
Spécialité : **INFORMATIQUE**

Présentée et soutenue publiquement par

Bruno PINAUD

le 22 mai 2006
à l'Ecole polytechnique de l'université de Nantes

Contribution à la visualisation des connaissances par des graphes dans une mémoire d'entreprise : application sur le serveur Atanor

Jury

Rapporteurs :	Jin-Kao Hao, Professeur à la faculté des sciences d'Angers,
	Gilles Venturini, Professeur à l'Ecole polytechnique de l'université de Tours,
Examinateurs :	Henri Briand, Professeur à l'Ecole polytechnique de l'université de Nantes,
	Pascale Kuntz, Professeur à l'Ecole polytechnique de l'université de Nantes,
	Christel Vrain, Professeur à la faculté des sciences d'Orléans,
Invités :	Jean-Bernard Fournier, Président du directoire de PerformanSe Corp.,
	Jacques Philippé, Expert scientifique pour PerformanSe Corp.

Directeur de thèse : Henri Briand
Co-encadrante : Pascale Kuntz
Laboratoire : Laboratoire d'Informatique de Nantes Atlantique
2, rue de la Houssinière – BP 92208
44322 Nantes CEDEX 3.

Remerciements

Cette thèse a été réalisée grâce à la coopération du *Laboratoire d'Informatique de Nantes Atlantique* (LINA), de l'*Ecole polytechnique de l'université de Nantes* (Polytech'Nantes), et de la société *KnoweSia SAS* filiale de *PerformanSe Corp.* (Carquefou-44). Je tiens tout d'abord à remercier Monsieur Henri Briand pour avoir accepté d'être mon directeur de thèse. Je tiens aussi à remercier vivement Madame Pascale Kuntz pour sa très grande patience et son importante implication dans mon travail par un encadrement de tous les instants. Je remercie aussi Messieurs Jean-Bernard Fournier, Patrick Brunet, Serge Baquedano et Jacques Philippé pour m'avoir accueilli au sein de PerformanSe et m'avoir fait confiance. Je remercie aussi l'ensemble de mes collègues de KnoweSia et PerformanSe, ainsi que l'ensemble du personnel du département informatique de Polytech'Nantes pour l'ambiance de travail très chaleureuse et la convivialité. Je remercie aussi vivement Madame Catherine Galais pour ses nombreuses relectures pointues de publications anglophones et sa bonne humeur permanente.

Je tiens aussi à remercier ma famille et mes amis les plus proches notamment Vincent Geoffray, Vincent Fagard, Grégory Fernandez, Marina Goureman et William Dupuy pour leur soutien, leur aide et les nombreux bons moments passés tous ensemble. Je remercie aussi les membres de l'équipe COD pour leur soutien et leur aide notamment Julien Blanchard, Jérôme David ainsi que Rémi Lehn sans qui tout ce travail n'aurait pas été possible.

Je remercie aussi Messieurs Gille Venturini et Jin-Kao Hao d'avoir accepté d'être les rapporteurs de ma thèse, pour l'attention avec laquelle ils l'ont lue et évaluée ainsi que pour les remarques et critiques constructives qu'ils m'ont adressées.

Table des matières

Table des figures

Liste des tableaux

Liste des Algorithmes

Chapitre 1

Introduction

Cette thèse a été menée dans le cadre d'une convention *CIFRE* établissant une collaboration entre la société *KnoweSia SAS* (Carquefou, Loire-Atlantique), spécialisée dans le domaine de la gestion et de la valorisation des connaissances, et l'équipe *COD* (COnnaissances & Décision) du *LINA* (Laboratoire d'Informatique de Nantes Atlantique).

La Gestion des Connaissances

Parmi les problèmes stratégiques actuels auxquels sont confrontées les entreprises, on peut notamment citer la gestion de données distribuées dont les volumes ont explosé ou encore la baisse du niveau d'expertise dans certains domaines causée par l'érosion du nombre d'experts (pyramide des âges qui vieillit, mobilité importante). Dans un contexte fortement concurrentiel où la durée de vie de la plupart des produits diminue constamment et les technologies et méthodes utilisées pour la fabrication et la distribution évoluent sans cesse, il est important de pouvoir conserver en vue d'une réutilisation les informations pertinentes et les connaissances acquises. Cette mémorisation structurée des connaissances est d'autant plus difficile à effectuer que d'une part, les structures des produits sont de plus en plus complexes (multiplicité des technologies et des acteurs, grand nombre de composants), et que d'autre part les caractéristiques des sources de connaissances relatives à ces systèmes sont de multiples sortes (individuelles, collectives, tacites, procédurales, ...).

En effet, en plus des connaissances explicites (courriers électroniques, procédures, notes de service, ...), il faut aussi prendre en compte tout le savoir implicite ou tacite (bonnes pratiques, savoir-faire, ...) ; c'est à dire l'ensemble des connaissances qui ne sont pas facilement formalisables avec des formes syntaxiques usuelles [2, 36]. Le capital d'une entreprise n'est pas seulement

composé de ses biens matériels et outils de production. Sans l'ensemble des savoir-faire, la performance de ces outils se restreint fortement. Il faut donc capitaliser cette connaissance tacite, c'est à dire la conserver et l'analyser pour la rendre accessible aux utilisateurs concernés, la faire évoluer et par ce biais faire ainsi évoluer l'entreprise.

La Gestion des Connaissances (GC) peut être précisément définie comme une approche globale et transversale qui regroupe l'ensemble des méthodes et techniques permettant de formaliser, partager, diffuser et enrichir le capital intellectuel de l'entreprise. L'objectif sous-jacent est de pouvoir fournir à chaque classe d'utilisateurs les connaissances pertinentes au bon moment. La gestion des connaissances est donc un processus anthropocentré qui regroupe un ensemble de tâches difficiles. Elle peut prendre plusieurs aspects au sein d'une organisation et être traitée en utilisant de multiples approches [36]. Le travail de cette thèse s'inscrit dans le cadre du développement d'un Système de Gestion des Connaissances (SGC) ou serveur de connaissances que l'on peut qualifier de mémoire d'entreprise. Le processus de création d'une telle mémoire est considéré comme le passage d'une mémoire de travail à une mémoire organisationnelle. Elle se définit comme un capital de connaissances accessible indépendamment des acteurs qui l'ont créée [109].

Il faut noter que l'utilisation de l'informatique n'est *a priori* pas obligatoire en GC. Mais, comme le font remarquer de plus en plus d'auteurs [2, 28, 106], cette dernière est amenée à jouer un rôle de plus en plus important. Au-delà des fonctionnalités qu'elle apporte (*e.g.* diffusion de l'information, utilisation d'éditeurs graphiques adaptés), l'utilisation de l'informatique permet de renforcer l'interactivité avec l'utilisateur qui peut ainsi être placé au cœur du processus de GC.

Cycle de vie d'une mémoire d'entreprise

Une démarche de gestion des connaissances est associée au cycle de vie d'une mémoire d'entreprise. Il peut être schématisé par un processus complexe qui intègre, entre la détection des besoins et l'utilisation, différentes phases [28] (figure 1.1). La détection des besoins permet de qualifier le projet en terme d'ambition, de limites, d'impact organisationnel, et de quantifier les moyens nécessaires à sa bonne réalisation. Ensuite, une fois recensées les sources de connaissances disponibles et valides qui peuvent être utilisées (documentations papiers, experts humains, bases de données, ...), plusieurs types de mémoires sont envisageables : *e.g.* une Gestion Electronique de Documents (GED), une mémoire à base de cas, une documentation technique intelligente [108] ou encore, comme ici, une mémoire à base de connaissances.

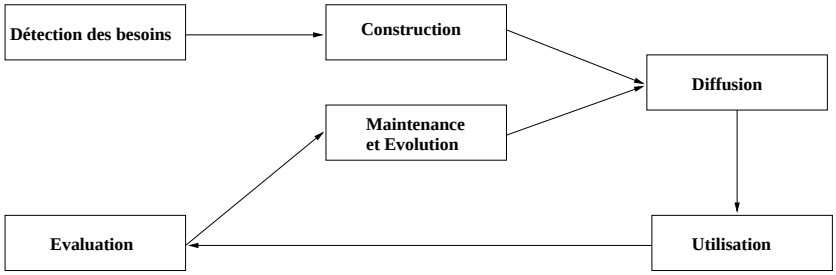


FIG. 1.1 – Cycle de vie d'une mémoire d'entreprise [28].

La construction d'une mémoire à base de connaissances nécessite la mise en place d'un formalisme précis pour représenter les connaissances (ontologies, modélisation objets, réseaux sémantiques, règles de productions, ...). Sur le plan opérationnel, ces représentations sont conçues à partir de méthodes d'ingénierie des connaissances. Citons par exemple la méthode CommonKADS [124] qui propose des modèles génériques permettant d'interpréter les données recueillies des experts ou encore la méthode MKSM[1] [40] qui consiste à construire un document, appelé *livre de connaissances,* qui contient les descriptions textuelles et graphiques des modèles de connaissances obtenus après extraction des expertises. Il existe de nombreuses méthodologies qui sont bien souvent spécialisées selon le secteur d'activité de l'entreprise. Chaque méthode utilise ses propres techniques de recueil de données parfois de façon complémentaire, comme l'entretien, l'observation directe de l'expert en situation de travail, ou encore l'utilisation d'éditeurs graphiques conçus spécifiquement. Ces éditeurs, utilisés directement par l'expert, facilitent la transmission directe des connaissances sur la base du formalisme de la mémoire.

A tout moment du cycle de vie, dès qu'il faut réaliser une interface entre le contenu de la mémoire et ses usagers, des représentations visuelles adaptées s'avèrent des médiateurs efficaces qui permettent de faciliter le dialogue [5, 23]. Ces représentations doivent être simples à appréhender et à comprendre pour les différentes classes d'utilisateurs. Elles doivent permettre d'apporter aux intervenants humains un substrat artificiel qui transcrive un grand nombre d'informations et qui soit un support à leurs connaissances et à leurs intuitions pour que non seulement ils puissent exprimer plus facilement leurs savoirs tacites et implicites mais aussi découvrir des nouvelles connaissances (*e.g.* relations).

[1] *Methodology for Knowledge System Management*

La visualisation en Gestion des Connaissances

Différents travaux en psychologie cognitive, en particulier issus de l'école de la Gestalt [77], ont mis en évidence la capacité du cerveau à analyser rapidement des composantes graphiques, et à pouvoir raisonner sur des représentations visuelles. C'est donc naturellement que conjointement à l'utilisation grandissante des Technologies de l'Information et de la Communication (TIC) s'est développée la visualisation d'information. La définition de Card *et al.* [18] est souvent prise comme référence : "*... the use of computer-supported, interactive, visual representations of abstract data to amplify cognition*". Mais comme le font remarquer Eppler et Burkhard [15, 39], cette définition atteint ses limites lorsque l'on veut traiter le cas spécifique de la visualisation de connaissances. En effet, les méthodes utilisées en visualisation d'information visent souvent la généricité et n'intègrent pas toujours la sémantique des données et leur domaine d'utilisation. De plus, ces méthodes peuvent être parfois difficiles à appréhender pour un utilisateur non initié. En GC, afin de faciliter les transferts de connaissances, il est nécessaire de développer des méthodes de visualisation qui permettent de respecter la sémantique des données et de masquer les différents aspects combinatoires et algorithmiques quand cela est nécessaire : *"Knowledge visualization examines the use of visual representations to improve the transfer of knowledge between at least two persons or group of persons"* (Burkhard [15]).

La visualisation d'information et la visualisation de connaissances exploitent donc toutes deux nos talents innés pour la manipulation de représentations graphiques de façon complémentaire : la première vise généralement à analyser de grandes quantités de données pour en faciliter la lecture et la compréhension, et la seconde vise à faciliter l'échange et la création de connaissances en mettant à disposition des outils pour permettre aux personnes de facilement exprimer et formaliser ce qu'elles savent [39]. La paire indissociable {modèle, représentation visuelle} dépend à la fois des connaissances dont on dispose, du mode de raisonnement sur ces connaissances et des différents points de vue "utilisateurs" considérés dans le SGC. D'une façon générale, la visualisation de connaissances est un domaine en plein essor, stimulé en particulier par les travaux sur les représentations visuelles du Web sémantique, et l'analyse de cet aspect fondamental dans un processus de GC n'en est qu'à ses débuts.

La très grande majorité des représentations visuelles actuelles proposées sont basées, au moins implicitement, d'un point de vue formel sur des modèles d'arbres ou plus généralement de graphes. De façon générale, les modèles de visualisation utilisés s'inspirent du modèle générique des réseaux séman-

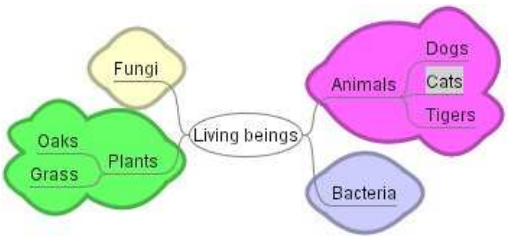


FIG. 1.2 – Une carte cognitive réalisée avec le logiciel Freemind.

tiques [85]. Dans ce modèle, les concepts sont représentés par les sommets d'un graphe et les relations sémantiques par les arêtes. La majeure partie des techniques présentées ci-dessous pourraient être considérées, dans le cadre d'une représentation descriptive, comme des spécialisations des réseaux sémantiques.

Représentation par arbres

Les représentations sous forme d'arbres, qui sont parmi les plus abouties, regroupent des techniques très différentes :

- Les cartes cognitives ont été développées pour faciliter la transcription d'idées sur un support visuel [16]. Autour d'un sommet central des "idées" sont représentées en créant différentes arborescences. A l'origine, les représentations étaient effectuées manuellement mais différentes solutions logicielles permettent d'enrichir la représentation. On peut notamment citer MindManager[2], VisualMind[3] et les différentes solutions proposées par "The Brain"[4], et dans le domaine du logiciel libre, FreeMind[5] qui est une solution très aboutie (figure 1.2).
- Les arbres de défaillance [88] sont principalement utilisés dans le domaine de la sûreté de fonctionnement. Ils permettent de représenter graphiquement l'ensemble des pannes d'un système qui peuvent se produire pour un évènement donné (figure 1.3). Un même système peut donc avoir plusieurs arbres de défaillance possibles. Ce mode de représentation est très employé dans le monde industriel quand la sécurité est primordiale (aéronautique, automobile, chimie, nucléaire, ...).
- Les arbres de décision, et plus généralement les graphes d'inductions, ont été initialement utilisés en apprentissage automatique [143] ; ils res-

[2] http://www.mmdfrance.fr/
[3] http://www.visual-mind.com/
[4] http://www.thebrain.com
[5] http://freemind.sourceforge.net/wiki/index.php/Main_Page

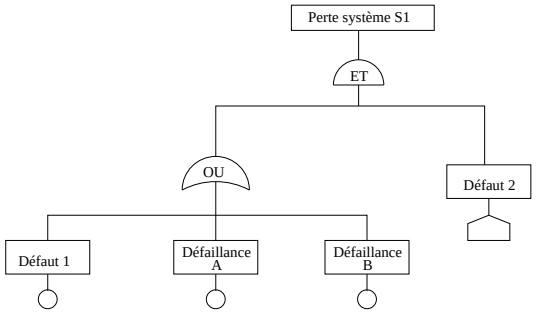


FIG. 1.3 – Un arbre de défaillance.

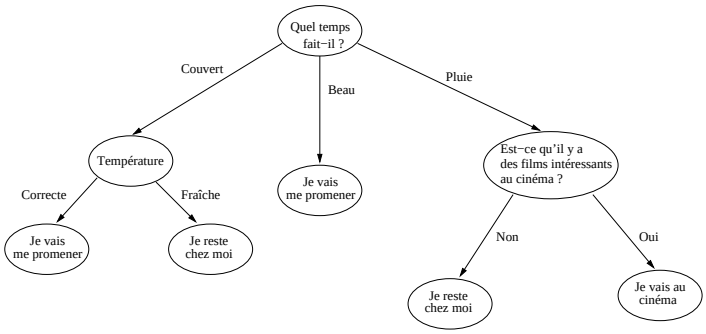


FIG. 1.4 – Un arbre de décision.

tent des modèles privilégiés d'exploration des données à la fois pour la description et le classement. Dans cette représentation, chaque nœud est associé à un test sur un attribut, les feuilles correspondant au résultat final et chaque arc est associé à une réponse possible d'un test (figure 1.4).

Représentation par graphes

La plupart des représentations par graphes en GC se retrouvent associées à trois grandes classes de modèles dont les intersections peuvent être importantes : les graphes conceptuels, les ontologies et les réseaux bayésiens.

– Les graphes conceptuels ont été à l'origine proposés comme une représentation graphique de la logique de premier ordre. Ils permettent de simplifier la mise en relation entre la logique et les langues naturelles [126] pour obtenir une représentation des données qui soit lisible et utilisable au sein de traitements automatiques. En GC, ils sont bien appropriés pour l'exploitation de relations entre les données [1],

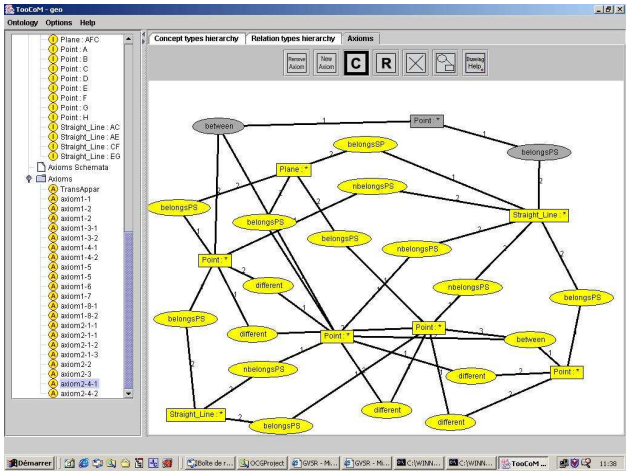


FIG. 1.5 – Un graphe conceptuel représentant une ontologie de la géométrie prospective réalisé avec TooCom [44].

et ils constituent un formalisme adapté pour la représentation d'ontologie [44] (figure 1.5).

- L'ingénierie ontologique vise à la construction et l'exploitation d'ontologies en conservant souvent une certaine indépendance par rapport aux usages opérationnels qui peuvent en être fait. Dans une mémoire à base de connaissances, les ontologies permettent d'expliciter la terminologie ou les concepts liés à un métier ou à un groupe donné d'individus au sein d'une organisation. Du fait de l'importance croissante en GC, différents logiciels proposent des représentations graphiques tels que Protégé[6], Os-Skill [117] ou encore ITM de Mondeca[7].
- Les réseaux bayésiens, utilisés initialement en apprentissage automatique, peuvent s'interpréter comme des graphes d'états auxquels sont ajoutés des probabilités de transition sur les arcs. Des logiciels sont maintenant utilisés en GC : BayesiaLab et Best[8].
- Des représentations graphiques spécifiques ont été développées pour des méthodes classiques de GC tel que CommonKADS [24].

[6] http://protege.stanford.edu/index.html
[7] http://www.mondeca.com/
[8] http://www.bayesia.com/index_fr.php

Contexte de la thèse

L'origine de cette thèse relève d'une problématique de visualisation associée à un SGC. Le serveur de connaissances Atanor développé par la société *KnoweSia* est une mémoire métier procédurale. Il a été conçu afin de gérer une démarche complète de gestion des connaissances en fusionnant l'expertise technique et les différentes ressources humaines disponibles. Cette démarche est orientée vers le déploiement et l'opérationnalisation des connaissances portant sur des systèmes complexes à partir de sources multiples [54, 55]. Atanor permet de rendre accessible la connaissance directement sur les postes de travail des utilisateurs finaux. Ils se retrouvent donc au centre de leur processus d'acquisition en naviguant directement au sein des connaissances qu'ils désirent utiliser. Ces connaissances maintenues par l'outil sont activables et présentées sous une forme multimédia [107]. L'architecture d'Atanor est fondée sur trois modules principaux qui proposent différentes vues sur les modèles de connaissances utilisés :

- le module *Expert* est destiné aux experts pour construire et faire évoluer les modèles de connaissances ;
- les utilisateurs ont à leur disposition le module *Praticien* pour, dans un contexte opérationnel, "activer" la connaissance ;
- le module *Manager* permet d'accéder à des indices structurels et opérationnels sur l'utilisation du serveur et des connaissances qu'il contient.

Chaque modèle nécessite une représentation visuelle spécifique adaptée aux besoins propres des utilisateurs concernés. Dans cette thèse nous nous focalisons sur le module *Expert*.

Un premier modèle visuel basé sur une extension des arbres de défaillance et des arbres de décision avait été proposé à l'origine du développement d'Atanor [55]. Cependant, son instanciation dans différents contextes applicatifs a mis en évidence différentes limites, la principale étant la présence de nombreuses redondances qui peuvent entraver l'interprétation synthétique du fonctionnement d'un processus et masquer des points critiques. L'objectif applicatif de cette thèse a été de proposer un nouveau modèle de représentation pour le module *Expert* d'Atanor ainsi que les algorithmes nécessaires à sa représentation visuelle.

Contribution et organisation de la thèse

La structure actuelle des connaissances du serveur Atanor nous a conduit à privilégier une représentation par graphes, plus précisément sous la forme de graphes en niveaux où les sommets –représentant les tâches– sont or-

donnés dans des niveaux –représentant les différentes étapes du processus au sens de l'expert–. L'implémentation de cette représentation nécessite la mise en œuvre d'une méthodologie proposant à l'utilisateur un tracé lisible et intelligible.

Le chapitre 2 présente une étude bibliographique orientée vers les usages des différents problèmes de représentation visuelle des graphes : le problème classique du tracé statique sur un support de taille standard, le problème dynamique où le graphe à tracer évolue sur une échelle de temps restreinte, le tracé des graphes de grandes tailles, les problèmes émergents des tracés en trois dimensions ainsi que les représentations basées sur d'autres codages des graphes. Nous présentons aussi les langages de description de graphes les plus utilisés dans les logiciels. Nous rappelons dans ce chapitre que la génération d'un "bon tracé" est souvent difficile en terme de complexité, et les critères utilisés pour estimer la qualité des tracés sont souvent subjectifs voire contradictoires puisqu'ils sont liés à la perception du tracé par l'utilisateur.

Ces critères se traduisent par des contraintes de lisibilité à satisfaire modélisées comme des contraintes d'optimisations [27]. Parmi les nombreuses contraintes, différents travaux ont montré l'importance en terme de lisibilité et de mémorisation de la minimisation du nombre de croisements d'arcs [110]. Nous nous sommes donc focalisés sur celle-ci. Pour les tracés d'un graphe en niveaux, la réduction du nombre de croisements d'arcs consiste à trouver un ordre optimal des sommets dans chacun des niveaux. Le chapitre 3 présente ce problème d'optimisation. L'étude bibliographique des différentes méthodes de résolutions connues montre que l'analyse des paysages de recherche a souvent été négligée. Cette étude est fondamentale afin de concevoir une méthode de résolution efficace. Nous montrons par une étude descriptive et statistique des paysages de recherche associés à différentes tailles de graphes qu'ils sont fortement multimodaux et que la qualité des optima locaux est très variable. Nous en concluons qu'un algorithme évolutionnaire et particulièrement un algorithme génétique (AG) semble être une voie prometteuse.

Le chapitre 4 présente une description complète, ainsi que la validation des différents opérateurs de l'algorithme génétique que nous avons développé. Il possède deux particularités : deux opérateurs de croisement spécifiques aux tracés en niveaux et une hybridation par une recherche locale. Les expérimentations montrent que cet algorithme surclasse les méthodes exactes classiques et donne dans la plupart des cas de meilleurs résultats en terme de qualité et plus rapidement que les autres métaheuristiques développées à notre connaissance pour ce problème.

Le chapitre 5 présente deux extensions de l'AG. La première est consacrée au problème de tracé dynamique qui doit tenir compte simultanément de deux contraintes : maintenir la lisibilité du tracé à chaque étape, ainsi

qu'une certaine persistance de la représentation dans le temps afin de limiter les efforts de réinterprétation de l'utilisateur. Nous proposons une heuristique pour ce problème. La deuxième extension dépasse le cadre applicatif et porte sur la comparaison de l'AG avec des méthodes de descentes multiples pour des graphes de grandes tailles qui ne peuvent plus être tracés sur des supports de taille standard. L'objectif initial de cette comparaison était de mieux comprendre les comportements respectifs de l'AG et des descentes multiples pour des corpus d'instances variés. Nous montrons sur des jeux de données significatifs que la performance de l'AG diminue au-delà d'une certaine taille des graphes à cause probablement de changements importants dans la structure des espaces de recherche.

Le chapitre 6 est consacré à l'application sur Atanor. Ses caractéristiques principales et le modèle actuellement utilisé pour la représentation des connaissances sont présentés. En utilisant un exemple réel d'application, rencontré dans un projet sur la maintenance de machines de tri automatique de courrier de la Poste, nous comparons les modèles de représentations. Nous montrons l'intérêt de la visualisation des connaissances par notre modèle de graphes pour l'amélioration de la lisibilité des tracés ainsi que leur exploitation par les différentes classes d'utilisateurs.

Chapitre 2

Représentation visuelle des graphes

Il faut imaginer dans sa tête des trucs qu'on appelle sommets, et pour toute paire de sommets soit une arête qui les joint, soit une non-arête qui les laisse sans joint : ceci est un graphe selon Berge.

P. Rosenstiehl [118]

Sommaire

Une conséquence de la définition d'un graphe donnée en exergue par l'un des plus célèbres maîtres du domaine est qu'ils sont les outils privilégiés pour simultanément modéliser et représenter visuellement un système de relations entre des entités. *"Quand on étudie la théorie des graphes, on pourrait très bien parler de graphe en terme de fonction en 0 et 1 (. . .), mais non, on les traite en forme de figure parce qu'on veut visualiser l'objet, mettre des points*

pour représenter des sommets ; les arêtes ce sont des lignes continues qu'on dessine sur le plan et ce sont des propriétés d'un type graphique et visuel qu'on étudie (...)[1]". Un des intérêts majeurs des graphes est effectivement de pouvoir caractériser les propriétés d'un système de relations via un arsenal combinatoire sophistiqué [9] tout en facilitant l'accès à ces structures complexes via notamment des représentations visuelles adaptées. Berge proposait de voir le dessin dans sa tête ! Mais heureusement, depuis les premiers travaux de Knuth [75] dans les années 60, la représentation visuelle des graphes sur des supports plus partageables est devenue un domaine de recherche à part entière, animé par la communauté *"Graph Drawing"*[2]. L'intérêt croissant ces dernières années pour ce domaine est stimulé par les applications où, dans de nombreux domaines (e.g. Gestion des Connaissances, Toile, Réseaux sociaux, Réseaux sémantiques, ...) les réseaux rencontrés ne cessent de se complexifier ne pouvant plus être traités aisément "à la main".

Il est souvent plus facile de justifier du recours à un modèle de graphe dans un cadre applicatif que de développer, et même plus simplement de choisir une méthode de tracé adaptée à sa problématique parmi les nombreuses méthodes et techniques existantes. Comme le notent Mutzel et Jünger [103] dans un ouvrage collectif récent consacré aux logiciels de tracés, la complexité de mise au point d'outils efficaces de visualisation est généralement sous-estimée par les novices ; les utilisateurs essayent souvent de développer des solutions par eux-mêmes qui, *in fine*, ne répondent guère à leurs véritables besoins. Une part de la complexité réside dans la nécessité de maîtriser des résultats provenant à la fois des mathématiques (essentiellement théorie des graphes, optimisation combinatoire, géométrie algorithmique) et de l'informatique (essentiellement algorithmique et structures de données, mais aussi génie logiciel et interfaces homme-machine).

Ce chapitre, restreint aux usages, présente un panorama synthétique des principales problématiques dont relève une démarche de représentation visuelle de graphes. Pour de plus amples détails sur les algorithmes mis en œuvre et leurs implémentations, nous renvoyons à trois ouvrages récents de référence : *Graph Drawing—Algorithms for the Visualization of Graphs* de Di-Battista *et al.* [27], *Drawing Graphs—Methods and Models* sous la direction de Kaufmann et Wagner [71] et *Graph Drawing Software* sous la direction de Mutzel et Jünger [103]. Les représentations en niveaux qui sont utilisées dans la suite de la thèse, sont détaillées spécifiquement dans le chapitre 3.

[1] citation extraite de `http://perso.wanadoo.fr/jacques.nimier/entretien_berge.htm`

[2] Un symposium international sur le tracé de graphes (*Int. Symp. on Graph Drawing*) est organisé annuellement avec des actes publiés sous la forme de *Lecture Notes in Computer Science* chez *Springer*.

Les différentes parties de ce chapitre sont organisées selon l'ordre des graphes à manipuler et les objectifs visés par la représentation visuelle. Le paragraphe 2.1 rappelle les problèmes de base soulevés par la représentation visuelle des graphes en les illustrant sur le problème du tracé statique. Les paragraphes suivants sont dédiés à des extensions du tracé statique en plein essor :

- le paragraphe 2.2 traite du tracé dynamique où le graphe évolue dans le temps ;
- le paragraphe 2.3 présente le tracé des grands graphes, dont l'ordre peut atteindre plusieurs milliers de sommets. Cette problématique connaît un intérêt croissant dû à l'explosion des quantités de données à manipuler dans de nombreux domaines ;
- le paragraphe 2.4 est consacré aux représentations tridimensionnelles qui ont émergé ces dernières années.

Au delà de ces représentations que l'on peut qualifier de classiques puisqu'elles représentent de façon explicite les arêtes des graphes par des courbes, il existe aussi d'autres méthodes qui utilisent des codages particuliers ou des métaphores graphiques pour représenter les graphes. Certaines sont présentées dans le paragraphe 2.5.

Différents langages de description ont été proposés pour coder les graphes et faciliter leur manipulation. Les principaux langages employés sont décrits dans le paragraphe 2.6.

2.1 Le tracé statique

Dans la suite nous considérons un graphe $G = (V, E)$ avec un ensemble V de sommets et un ensemble E d'arêtes associées à une relation binaire sur $V \times V$. Selon les cas, la relation peut être symétrique ou non (on parle alors d'arcs). Le problème générique de tracé le plus classique consiste à dessiner G sous une forme "intelligible" sur un support standard bidimensionnel [27]. L'utilisateur n'étant pas forcément un expert en combinatoire, la qualité du dessin est décisive pour l'appropriation de la représentation [111]. Pour préciser cette notion, qui reste *in fine* subjective, on retient généralement quatre concepts de base (Di-Battista *et al.* [27]) : la convention de tracé, les contraintes physiques, les critères esthétiques et les contraintes sémantiques.

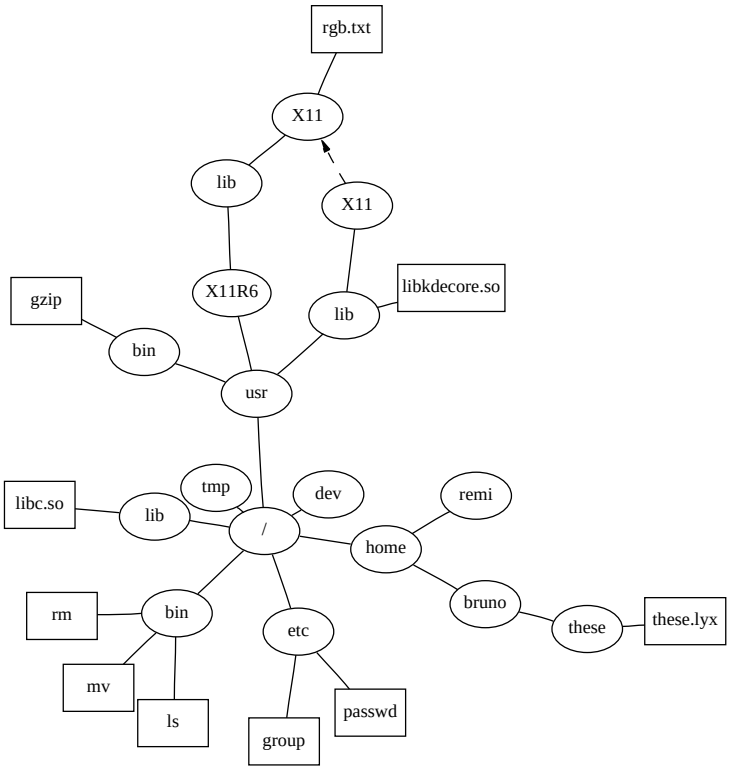
1. **La convention de tracé**
   Elle spécifie les règles géométriques de lecture du tracé qui sont souvent inhérentes aux pratiques en vigueur dans le domaine d'application. La figure 2.1-a montre un extrait d'un système de fichiers sans utiliser la convention de tracé usuelle du domaine. Les chemins d'accès aux fichiers

sont difficiles à lire. En revanche, la figure 2.1-b utilise la représentation habituelle avec une arborescence et les chemins d'accès aux fichiers sont plus simples à lire. La figure 2.2 montre différents exemples de tracés d'un graphe G associés à différentes conventions : (a) représentation polygonale où chaque arc est représenté par une ligne polygonale, (b) représentation rectiligne où chaque arc est représenté par un segment de droite, (c) représentation orthogonale où les sommets sont placés aux intersections d'une grille et les arcs sont positionnés sur la grille et (d) représentation en niveaux ou hiérarchique où les sommets sont rangés dans des niveaux verticaux ou horizontaux et les sommets d'un même niveau ne sont pas reliés entre eux par des arcs.

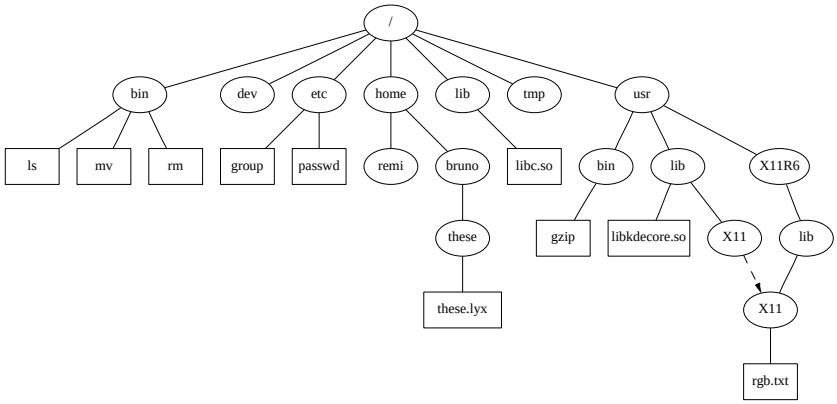
2. **Les contraintes physiques**
   Les contraintes du support de l'œil humain imposent notamment des écarts minimums à respecter entre les entités géométriques (points, boîtes, ...) représentant les sommets et les courbes représentant les arcs.

3. **Les critères esthétiques**
   Ils tentent de formaliser les propriétés à satisfaire pour faciliter la lisibilité d'un tracé. Ces critères sont définis par des contraintes combinatoires : minimisation du nombre de croisements d'arcs, minimisation de la somme des longueurs des arcs ou de l'arc de longueur maximale, minimisation des coudes dans les tracés orthogonaux, ... Il n'existe pas d'ordonnancement générique de ces critères ; l'interprétation de chaque tracé dépendant de sa propre sémantique et de la sensibilité de l'utilisateur. Cependant, des travaux en psychologie cognitive ont montré que la réduction des croisements est un des critères prépondérants pour la lisibilité et la mémorisation [110] (figure 2.3). Purchase [111] a aussi montré que les principales préférences portaient sur : la minimisation des croisements, la minimisation des coudes, le positionnement horizontal des étiquettes représentant les sommets et la jonction des arcs d'héritage (arcs de même origine). Plus récemment Ware *et al. [137]* ont montré, en s'appuyant sur les travaux de la théorie de la Gestalt [77], qu'une "bonne continuité" des arcs est nécessaire en plus de la réduction du nombre de croisements. En effet, lorsque la recherche de chemins est importante dans un graphe, la lecture est facilitée lorsque les angles formés par les arcs ne sont pas trop aigus (figure 2.4).

4. **Les contraintes sémantiques**
   Elles sont associées à l'interprétation des composantes du graphe ; par exemple, des proximités sémantiques doivent être respectées dans le positionnement des sommets.

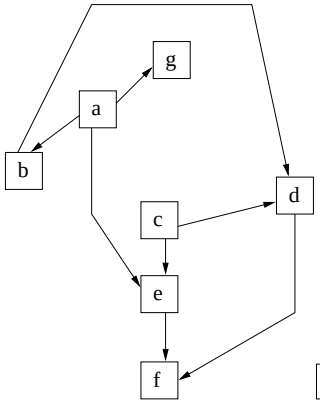


(a) Aucune convention particulière

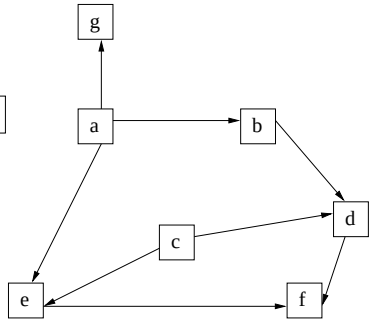


(b) La représentation arborescente conventionnelle du domaine.

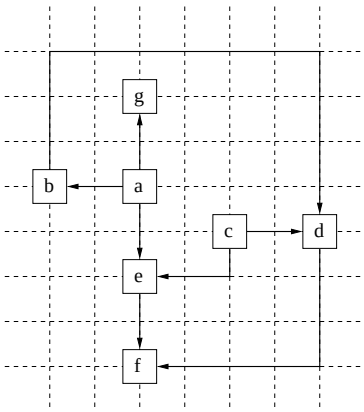
FIG. 2.1 – Deux représentations d'un même extrait d'un système de fichiers.



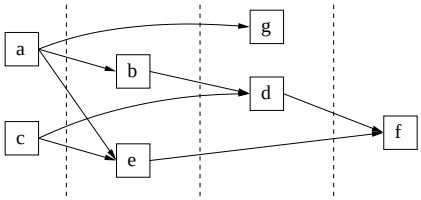
(a) Représentation polygonale



(b) Représentation rectiligne



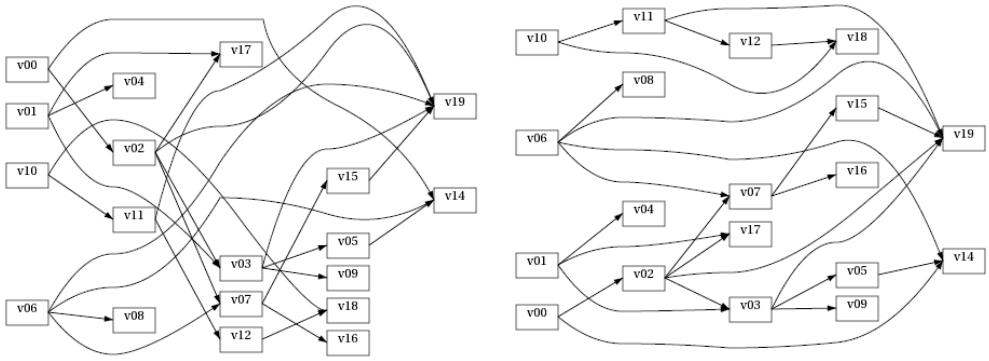
(c) Représentation hortogonale



(d) Représentation en niveaux

Soit un graphe G *avec* $V = \{a, b, c, d, e, f, g\}$ *et* $E = \{(a, b), (a, g), (a, e), (b, d), (c, d), (c, e), (d, f), (e, f)\}$.

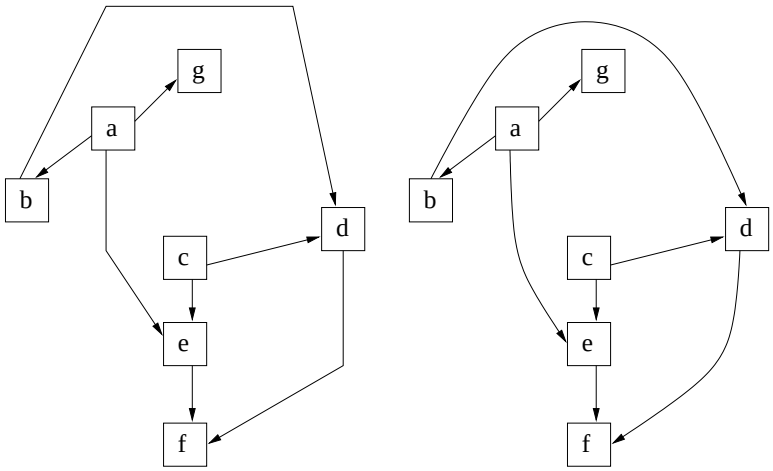
FIG. 2.2 – Différentes représentations d'un même graphe en utilisant plusieurs conventions de tracé.



(a) Pas d'optimisation du nombre de croisements d'arcs (37 croisements).

(b) Nombre de croisements réduit au maximum (5 croisements).

FIG. 2.3 – Illustration du respect des critères esthétiques.



(a) Représentation polygonale classique.

(b) Utilisation de lignes dont les angles sont aplanis pour permettre une meilleure lecture du tracé.

FIG. 2.4 – Illustration du principe de bonne continuité des arcs [137].

Lorsque la convention de tracé est fixée ainsi que les différentes contraintes, deux problèmes se posent généralement : le problème d'optimisation associé est bien souvent *NP*-difficile [48] et, lorsque l'on cherche à optimiser simultanément plusieurs critères esthétiques, les contraintes associées sont incompatibles. On doit alors avoir recours à des algorithmes approchés ou des heuristiques qui, basés sur des approches très différentes, ne conduisent pas nécessairement à des tracés identiques. Le choix de la méthode est alors un problème délicat pour l'utilisateur (de plus amples détails sur les méthodes, leurs choix et les contraintes associés sont donnés dans le chapitre 3 dans le cadre du tracé en niveaux).

De plus, les expérimentations (voir les challenges du *Symposium Graph Drawing*[3]) ont montré que les résultats des algorithmes automatiques ne devaient pas être systématiquement considérés comme les meilleures solutions pour un problème donné : ils peuvent être souvent améliorés *a posteriori* à la main. A la manière du tracé statique, les différents concepts énoncés servent de support aux autres types de tracés. Des contraintes supplémentaires sont ajoutées pour tenir compte des spécificités du tracé à produire.

2.2 Le tracé dynamique

Dans un contexte dynamique où les données à représenter évoluent sur une échelle de temps restreinte, des modifications du graphe telles que des ajouts ou retraits de sommets et/ou d'arêtes entraînent des changements visuels sur le tracé. La solution naïve qui consiste à reconsidérer à chaque instant t le problème comme un nouveau problème indépendant se heurte à deux écueils : *(i)* elle peut être inutilement coûteuse en temps de calcul en refaisant des calculs déjà faits en $t-1$, et *(ii)* elle peut perturber la carte mentale de l'utilisateur. En effet, contrairement au tracé statique où l'ensemble des contraintes liées au tracé sont connues à l'avance et ne changent pas dans le temps, le tracé dynamique doit prendre en compte le fait que l'utilisateur s'est déjà approprié le tracé présenté à l'instant précédent. Donc, en plus des contraintes énoncées précédemment pour le tracé statique qui doivent toujours être respectées, la lecture des dessins dans un contexte évolutif conduit à l'introduction de contraintes additionnelles. A chaque instant t, le nouveau tracé d'un graphe G_t doit également permettre une transition aisée pour l'utilisateur avec le tracé du graphe G_{t-1} présenté à l'instant précédent ; l'objectif étant de limiter l'effort cognitif nécessaire à l'interprétation des tracés successifs.

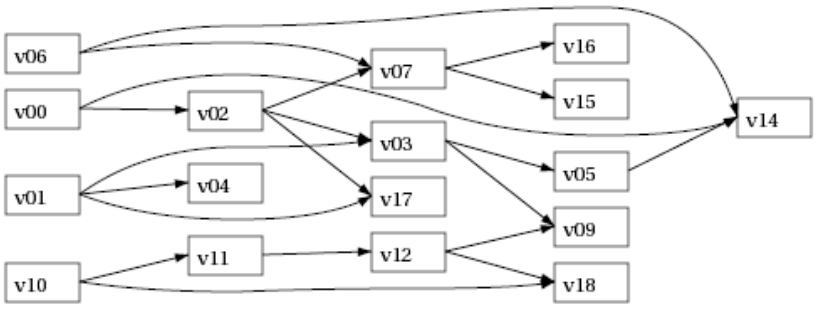
[3] Voir la partie *"GD Contest"* sur les différents sites de la conférence : `http://www.gd2005.org`, `http://www.gd2004.org` pour les deux dernières éditions.

Deux approches majeures sont proposées. La première consiste à mettre en évidence les changements *via* des animations graphiques. Elles doivent être suffisamment lentes pour permettre à l'utilisateur d'enregistrer les différentes modifications opérées sur le graphe [14, 25]. La seconde, et la plus populaire, consiste à préserver au mieux la stabilité des tracés en limitant les perturbations apportées sur le nouveau tracé par rapport aux précédents [22, 33]. La stabilité est une notion complexe qui dépend des caractéristiques géométriques et combinatoires du tracé, mais aussi des facultés de perception et de mémorisation de l'utilisateur. Cependant, deux facteurs prédominants semblent se dégager :

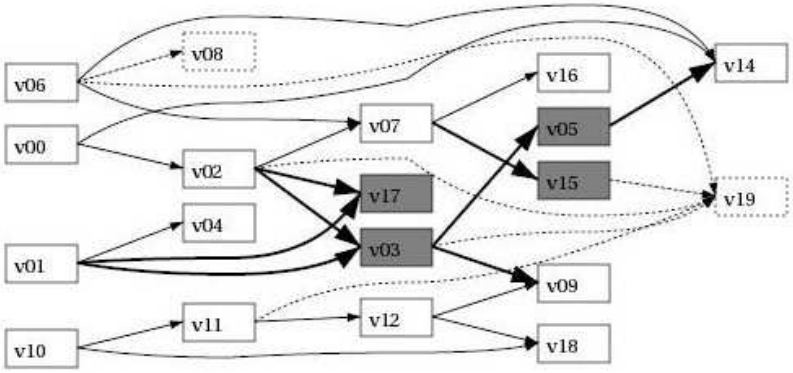
- les positions des sommets doivent changer le moins possible. Leur stabilité semble plus importante que celle des arêtes : les sommets servent de repères spatiaux alors que les arêtes sont essentiellement utilisées pour découvrir des relations entre des sommets déjà localisés ;
- l'ordre relatif des composantes du graphe dans le repère géométrique adopté doit être conservé tant que cela est possible car l'utilisateur repère les composants les uns par rapport aux autres.

Pour rendre opérationnelles ces contraintes de stabilité et ainsi, mesurer le degré de conservation de la carte mentale de l'utilisateur entre les tracés de deux graphes successifs, différentes métriques basées sur des indices de similarité ont été proposées [12]. La figure 2.5 illustre le principe de la prise en compte d'un critère de similarité dans le tracé d'un graphe en niveaux. Le critère est simplement basé sur la position des sommets dans chacun des niveaux. Le tracé de la figure 2.5-a est le tracé initial avec 2 croisements après optimisation. A $t+1$, l'utilisateur ajoute deux sommets et quelques arcs. Sur les figures 2.5-b et 2.5-c, les nouveaux sommets et les nouveaux arcs sont représentés avec des traits en pointillés. La figure 2.5-b représente le graphe à $t+1$ avec comme unique contrainte la réduction du nombre de croisements d'arcs ; il en reste ici 7. Malgré la taille réduite du graphe, des changements importants sont introduits et deux paires de sommets sont inversées (sommets à fond foncé) par rapport à la figure originale. De plus les différents arcs (dessinés avec des traits épais) associés à ces sommets ont aussi été déplacés. Pour palier à ce problème, le tracé de la figure 2.5-c tient compte du critère de similarité avec le premier tracé. On retrouve exactement le graphe d'origine mais le tracé comporte 9 croisements.

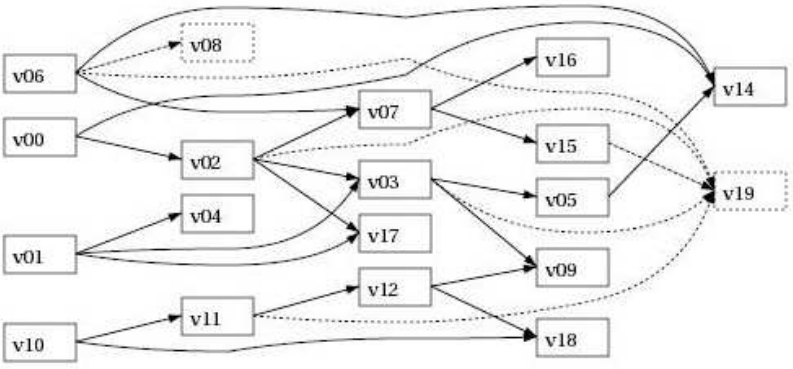
Comme pour tout problème multiobjectifs qui consiste à trouver un bon compromis entre les différents paramètres [20], la difficulté du tracé dynamique est qu'il faut à chaque étape trouver un tracé réalisant un bon compromis entre la conservation de la carte mentale de l'utilisateur, la lisibilité du tracé et le temps de calcul (les préconisations habituelles stipulent qu'il faut éviter d'avoir plus de 200ms entre deux tracés successifs si l'on veut



(a) Le graphe initial à l'instant t avec 2 croisements d'arcs.



(b) Le graphe à $t+1$ avec uniquement une optimisation du nombre de croisements. Les arcs en gras et les sommets foncés ont changé de place. Il reste 7 croisements d'arcs.



(c) Le graphe à $t+1$ en tenant compte d'un critère de similarité et d'une optimisation du nombre de croisements. Il reste 9 croisements.

Les sommets v08 et v19 ont été ajoutés au graphe après l'étape (a). Ces sommets et leurs arcs sont représentés en pointillés dans les figures (b) et (c).

FIG. 2.5 – Exemple de transitions entre un graphe G_t et G_{t+1} dans un contexte dynamique.

qu'un utilisateur ait une impression de fluidité entre les tracés [135]). Cependant, si des heuristiques ont été proposées pour les grandes familles de tracés (orthogonaux, en niveaux, ...), le problème reste encore largement ouvert et à notre connaissance peu de logiciels intègrent actuellement la gestion dynamique de façon efficace.

2.3 Le tracé des grands graphes

Dès que le graphe à représenter a des caractéristiques qui ne permettent plus une représentation exhaustive de chaque composante sur un support de taille standard comme un écran d'ordinateur, les algorithmes classiques de tracés de graphes ne sont plus applicables seuls. La plupart doivent être combinés avec des outils d'interaction qui diffèrent selon le type de graphes à représenter et il devient alors difficile d'avoir une méthodologie générique. Deux exemples permettent de concrétiser les nouveaux ordres de grandeur mis en jeu. Les graphes de contacts ou de co-citations dans les réseaux sociaux peuvent porter maintenant sur des centaines de milliers de sommets [131]. Pour estimer des paramètres décrivant la structure des relations sur la Toile, Reka *et al.* [112] travaillent sur des graphes échantillons de 300 000 documents et 1 500 000 liens (estimé à environ 0.3% de la Toile lors de la parution de l'article en 1999). En plus du problème d'affichage sur un écran statique de taille limitée, il faut également prendre en compte les problèmes aigus de gestion de la mémoire et du temps de calcul.

Les différentes méthodes mises en œuvre peuvent être classées en plusieurs catégories caractérisées par un ordre des graphes à tracer croissant : *(1)* application d'une distorsion sur les tracés comme le fameux *"fisheye"*, *(2)* utilisation d'une combinaison des stratégies de tracés de graphes et de recherche d'informations, *(3)* utilisation des propriétés spécifiques de la structure à tracer pour optimiser la place disponible, *(4)* affichage partiel du graphe.

2.3.1 Distorsion du tracé

Les premières techniques apparues sont basées sur un principe de distorsion des figures ; le but étant de fournir à l'utilisateur une vue unique de l'ensemble de la structure à tracer en appliquant diverses transformations. La technique des *"fisheyes"* [123] permet de visualiser des grandes structures de données que ce soit des arbres ou des graphes. Elle consiste à dessiner une partie de la structure le plus lisiblement possible, en se focalisant sur un sommet précis, appelé centre, et son voisinage immédiat puis à appliquer une distorsion sur l'affichage du reste de la structure pour offrir une vue macro-

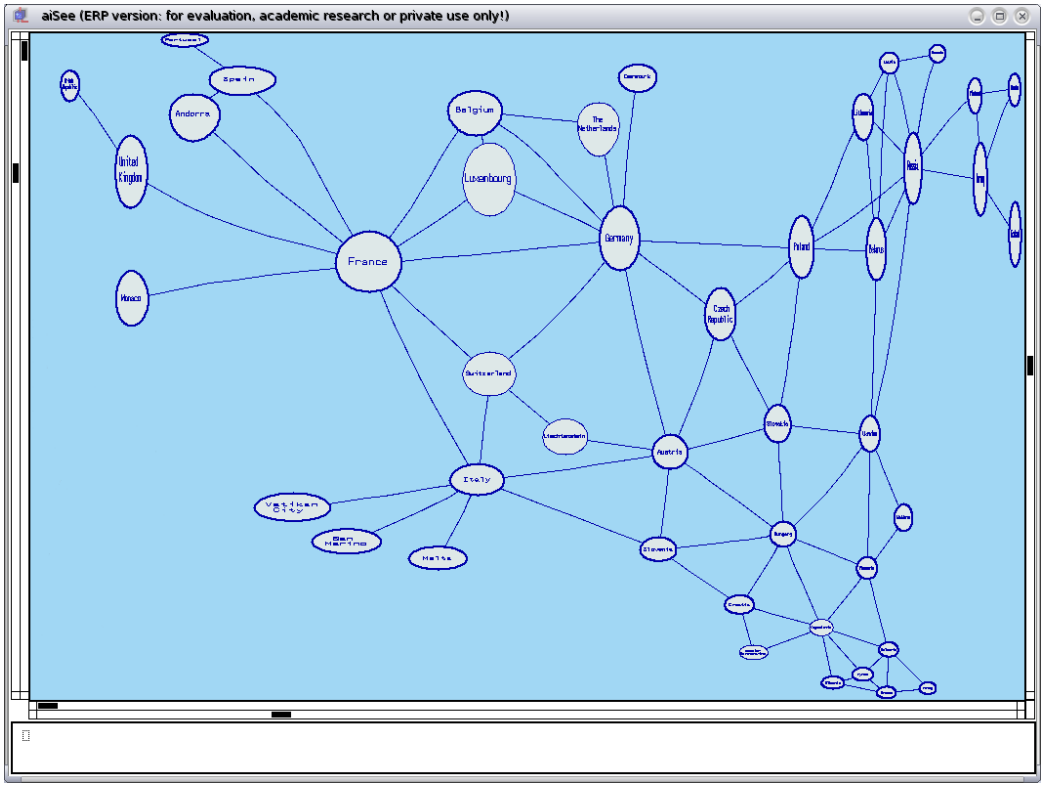
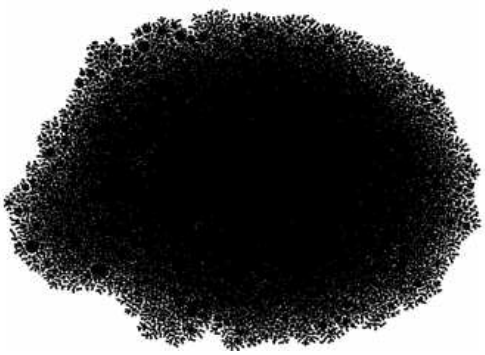


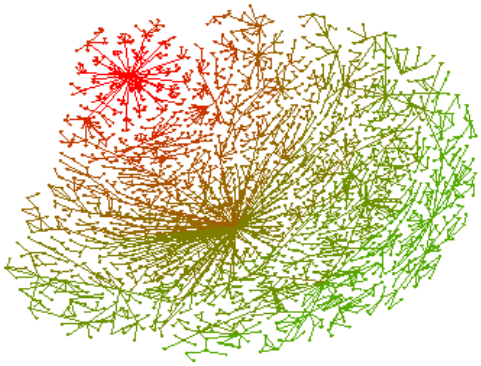
FIG. 2.6 – Un graphe qui représente les frontières communes entre les différents pays du continent européen. Dessin réalisé avec le logiciel Aisee `http://www.aisee.com`.

scopique. L'utilisateur a donc des informations sur la position du centre et de son voisinage par rapport à l'ensemble du graphe et peut exploiter au mieux les informations intéressantes pour lui sur la partie clairement dessinée (figure 2.6). L'inconvénient majeur de cette technique de représentation est que l'échelle des distances entre les sommets évolue. Plus on s'éloigne du centre, plus l'échelle est réduite de façon non linéaire et les distances compressées.

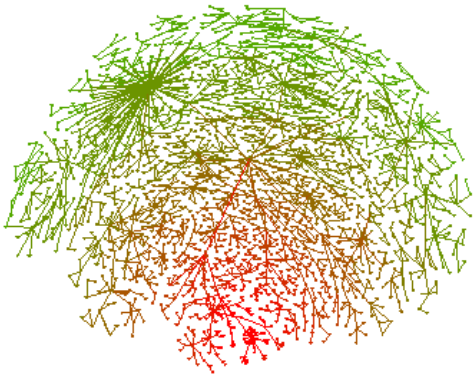
Récemment, Gansner *et al.* [46] ont proposé une méthode pour éviter ce problème de compression des distances. Au lieu de vouloir représenter l'ensemble de la structure, leur méthode simplifie la représentation en calculant une approximation des parties du graphe sur lesquelles l'utilisateur n'est pas focalisé. La figure 2.7-a représente une cartographie d'Internet qui est illisible si on veut directement tracer l'ensemble du graphe (87931 sommets et 87930 arêtes). Sur la figure 2.7-b, l'utilisateur se focalise sur la partie en haut à gauche de la figure dessinée en rouge. L'ensemble des sommets et arêtes de



(a) Graphe représenté en entier (87931 sommets et 87930 arêtes).



(b) Focus en haut à gauche de la figure.



(c) Focus sur le bas de la figure.

FIG. 2.7 – Différentes représentations d'une cartographie d'Internet en utilisant des "*Topological fisheyes*". Extrait de [46].

cette partie est complètement dessiné. Ensuite, en fonction de l'éloignement, les parties périphériques de la représentation sont de plus en plus simplifiées. Un dégradé des couleurs du rouge vers le vert permet d'indiquer le niveau de simplification. Cette technique permet aussi à l'utilisateur de naviguer dans le graphe en déplaçant la partie sur laquelle il se focalise. La simplification progressive de la structure permet d'aider à la conservation de la carte mentale de l'utilisateur. L'inconvénient principal est que pour calculer les différentes approximations, il est nécessaire de conserver en mémoire le graphe original.

2.3.2 Stratégie de recherche d'information

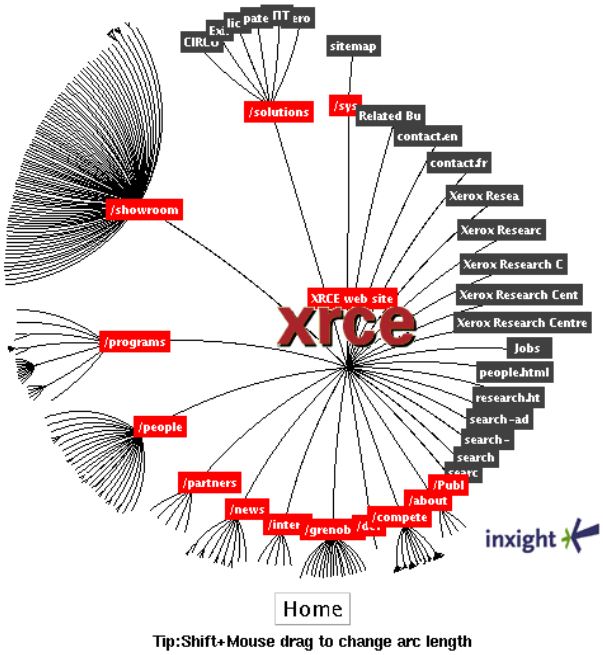
De plus en plus les méthodes classiques développées par la communauté *"Graph Drawing"* sont combinées avec des approches développées dans le contexte plus général de la visualisation d'information (voir la thèse de T. Munzner [102] qui propose une bibliographie pour la visualisation de grandes structures de données). Les approches proposées, développées récemment en commun par les deux communautés [59], intègrent plusieurs stratégies très différentes : distorsion de la représentation, partitionnement, navigation. La représentation exhaustive des composantes du graphe n'étant plus possible, le problème est souvent abordé comme un problème de recherche d'information : il s'agit de procurer à l'utilisateur une panoplie d'outils qui peuvent être combinés dans une stratégie de recherche [90]. La sélection des modes de représentation est souvent laissée à l'utilisateur ; la recherche actuelle est plutôt axée sur le développement de nouvelles méthodes capables de traiter des tailles croissantes. Bien que la question de la comparaison, en terme d'usage, soit de plus en plus discutée au sein notamment de la communauté "*InfoViz*", les analyses de retour d'expérience restent encore très limitées [10].

Une autre grande famille de méthodes se réfère explicitement à la stratégie proposée en recherche d'informations par Shneiderman [125] : *"Overview first, zoom and filter, then details-on-demand"*. La démarche est alors la suivante : *(i)* proposer à l'utilisateur une vue macroscopique de la structure, *(ii)* lui permettre de se focaliser sur un sous-graphe –par clustering [8] ou fragmentation [4]–, et *(iii)* rendre accessible les caractéristiques précises d'une donnée sélectionnée. Ce processus itératif s'arrête lorsque l'utilisateur a acquis suffisamment d'informations pour répondre à des besoins non nécessairement spécifiés préalablement. Une des difficultés est ici d'appliquer des transformations qui restent compréhensibles par l'utilisateur pour éviter de trop perturber sa carte mentale.

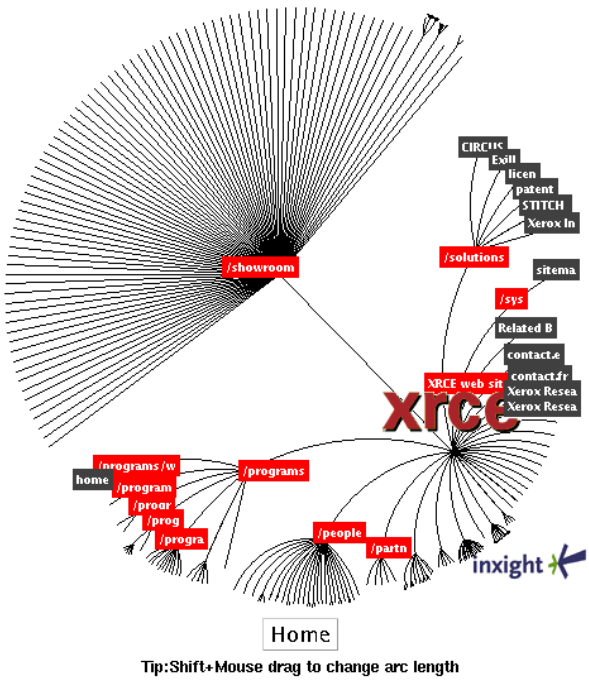
2.3.3 Projections dans d'autres espaces

Pour des graphes dont l'ordre peut atteindre plusieurs milliers de sommets, les méthodes de tracés génériques qui plongent le graphe dans un espace euclidien deviennent difficilement exploitables. Ces méthodes doivent être spécialisées pour utiliser au maximum les propriétés de la structure à dessiner et optimiser la place disponible pour le tracé. Par exemple, pour les arbres, *Xerox Research* a mis au point une technique qui utilise un espace hyberbolique pour représenter des arbres ; ce sont les arbres hyperboliques (figure 2.8)[4]. Avec cette technique, proche du *"fisheye"*, l'arbre est dessiné

[4]Technique commercialisée par la société Inxight, `http://www.inxight.com`



(a) L'arbre d'origine.



(b) L'arbre après un déplacement du centre.

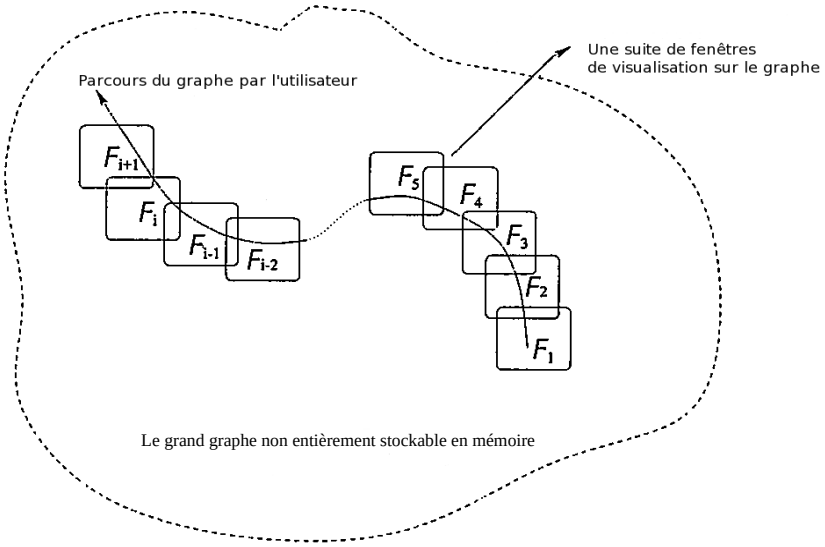
FIG. 2.8 – Deux vues du site web de "*Xerox Research Centre Europe*" représentées avec des arbres hyperboliques. Extrait de `http://www.xrce.xerox.com/sys/htree/`.

dans un espace hyperbolique et l'utilisateur se focalise sur un sommet et son voisinage. Plus on s'éloigne de ce sommet, moins il y a de détails sur la figure (figure 2.8-a, le centre de la figure est symbolisé par le nœud appelé *xrce*). La navigation s'effectue en déplaçant les sommets. La partie clairement dessinée se situe au centre de la figure. Un point important de cette représentation est que l'utilisateur sait toujours où se situe le centre de l'arbre qu'il visualise (figure 2.8-b). En revanche, les parties de la figure trop éloignées du sommet sur lequel est focalisé l'utilisateur peuvent ne pas apparaître.

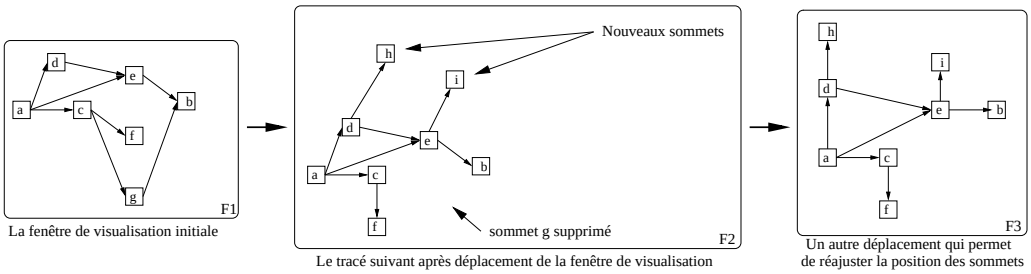
2.3.4 Affichage partiel

Sur des graphes dont l'ordre est encore plus important -au delà de 100 000 sommets-, la complexité des méthodes précédentes ne permet plus leur application sur l'intégralité de la structure. De part l'importance de sa taille, cette structure peut ne pas être entièrement stockée en mémoire lors de l'initialisation des algorithmes de tracé. L'approche *"Online Graph Drawing"* [63] permet dans ce cas à l'utilisateur de se concentrer dans un premier temps sur un sous-graphe, ou fenêtre de visualisation, qui peut être entièrement montré sur son écran en utilisant seulement les techniques du tracé statique (*logical frame*, le rectangle noté F_1 sur la figure 2.9-a). Ensuite l'utilisateur peut faire glisser cette fenêtre pour visualiser la suite du graphe (les différentes fenêtres F_i sur la figure 2.9-a). Au fur et à mesure de l'exploration, les parties manquantes sont calculées et stockées en mémoire. Ces différentes fenêtres de visualisation respectent les contraintes du tracé statique (figure 2.9-b). Les transitions entre les fenêtres sont conçues pour préserver au mieux la carte mentale de l'utilisateur en ne changeant que quelques sommets lors du passage d'une fenêtre à la suivante ou en effectuant un déplacement des sommets pour préparer la transition vers une autre fenêtre. On retrouve une problématique proche du tracé dynamique.

D'autres approches utilisent des affichages incrémentaux. Soit en montrant initialement les composantes les plus importantes puis en complétant peu à peu la représentation selon les désirs de l'utilisateur [140], soit comme dans Tulip [3], en définissant préalablement une métrique sur le graphe qui permet d'afficher de façon incrémentale un tracé visuellement proche du tracé complet avec un nombre très réduit d'éléments. La figure 2.10 illustre cette dernière technique en montrant l'affichage en 4 étapes d'un graphe d'ordre restreint qui représentent la structure d'un site web. Il faut noter que la première étape de la représentation qui contient moins de 6.5% des sommets de l'image finale est en fait "le cœur du site", c'est à dire les pages principales.

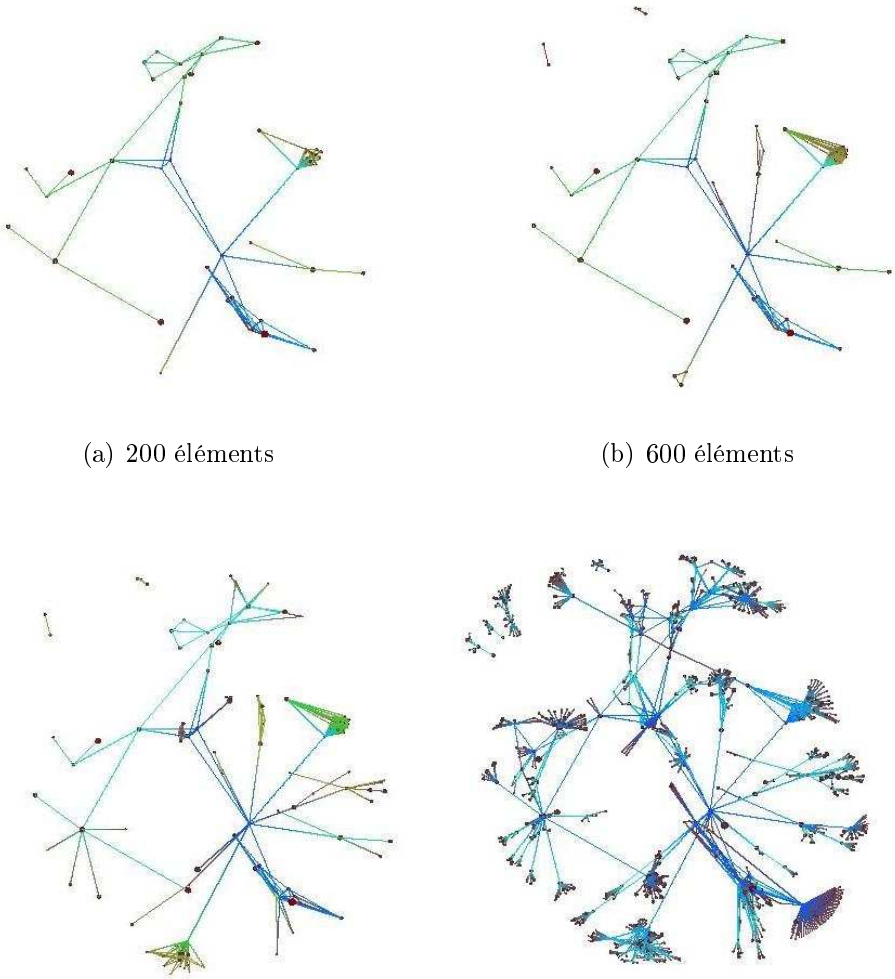


(a) Principe général de la méthode



(b) Extrait des représentations partielles successives.

FIG. 2.9 – Illustrations de la méthode "*Online graph drawing*". Adaptées depuis [63].



(a) 200 éléments (b) 600 éléments

(c) 1000 éléments (d) 3200 éléments

FIG. 2.10 – Affichage incrémental de la structure d'un site web avec Tulip. Extrait de [4].

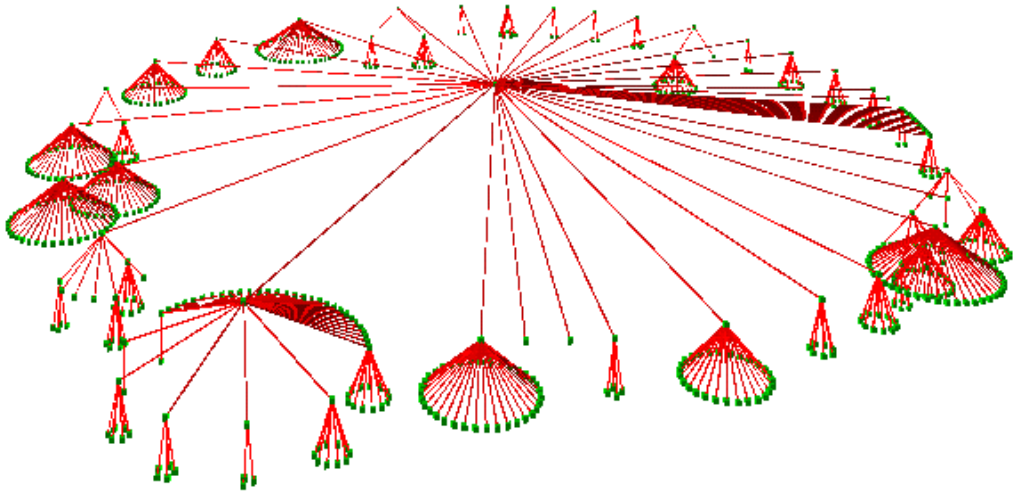


FIG. 2.11 – Un "*cone trees*" pour représenter un système de fichiers Unix.

2.4 Tracé tridimensionnel

Depuis les premiers prototypes des années 90 [58, 127] et l'exemple bien connus des *"cone-trees"* [116] (figure 2.11), l'accessibilité croissante à la fois des langages de développement pour des environnements tri-dimensionnels (*e.g.* VRML, *Virtual Reality Modeling Language*)[5] et des moyens de calculs disponibles, conduit à un intérêt grandissant pour les représentations visuelles des graphes en 3D [84], en particulier pour les structures de grande taille. L'ouvrage *"Atlas of Cyberspace*[6]*"*, limité aux représentations associées à la toile [31], et le site *"Visual Complexity*[7]*"* donnent un large aperçu des possibilités des cartographies tri-dimensionnelles.

Les travaux sur l'appropriation humaine de cette nouvelle approche n'en sont qu'à leur début [115, 136]. Quand elle est utilisée à bon escient, les intérêts avancés sont souvent que la dimension supplémentaire permet un ajout d'informations et une plus grande flexibilité pour placer les sommets et les arcs [56, 72]. Cependant, les restitutions actuelles sur les supports classiques restent bidimensionnelles, et par conséquent les tracés résultant peuvent être complexes et difficiles à appréhender. Pour éviter l'utilisation d'algorithmes spécifiques souvent complexes, les données à représenter doivent se prêter

[5] `http://www.graphcomp.com/info/specs/vrml10.html`

[6] Voir aussi le site `http://www.cybergeography.org/atlas/atlas.html` associé à cet ouvrage qui contient des représentations plus récentes.

[7] `http://www.visualcomplexity.com/vc/index.cfm`

"naturellement" à une représentation en 3D. L'ajout de la nouvelle dimension doit réellement apporter des informations supplémentaires à l'utilisateur et non pas une complexité accrue du tracé.

Un problème récurrent, bien connu des graphistes, dû à l'ajout de la troisième dimension, est celui de l'occlusion [135]. Ce phénomène se produit de façon générale lorsque un objet se retrouve masqué par un autre (voir l'exemple du haut de la figure 2.11). L'occlusion peut prendre diverses formes pour le tracé de graphes et peut, selon l'angle de vision de l'utilisateur, modifier la représentation [84] : deux sommets peuvent être vus comme un seul, des croisements supplémentaires inexistants dans la représentation apparaissent, des sommets et des arcs peuvent se superposer et créer de nouveaux sommets, ... Sur le plan théorique, les projections utilisées doivent limiter la perte d'information. Pour faire face à ces limitations, les logiciels intègrent des outils de navigation de type rotation, déplacement, zoom qui augmentent l'efficacité de l'utilisation de l'espace de représentation.

Pour les structures de grande taille, une extension du modèle des graphes hyperboliques présenté précédemment a été développée pour une famille de graphes peu denses [101]. L'espace hyperbolique 3D permet à l'utilisateur de se focaliser sur un sommet particulier tout en conservant une vue globale de l'ensemble de la structure (figure 2.12).

2.5 Au delà des représentations sommets-liens

Au-delà des représentations visuelles classiques des graphes, où les sommets et les arcs sont explicitement tracés, se sont développées ces dernières années, essentiellement pour les arbres, des nouvelles approches basées sur des métaphores graphiques ; citons par exemple celle du circuit électronique (figure 2.13) ou de la métaphore botanique [74] (figure 2.14).

Pour tenter de braver la complexité inhérente aux grands tracés, d'autres modes de représentation ont été proposés. Ils peuvent être basés par exemple sur une représentation matricielle comme dans les travaux de Ghoniem *et al.* [49], ou encore, pour les arbres, par un pavage du plan avec la méthode des Treemap [64] (figure 2.15). L'avantage principal de cette méthode populaire est que des arbres de plusieurs centaines de sommets peuvent être représentés sur un support de petite taille tout en restant lisibles et compréhensibles pour un utilisateur ayant intégré le mode de lecture.

Ces méthodes, par rapport aux représentations classiques des graphes, ont l'inconvénient de nécessiter pour l'utilisateur une période d'appropriation du codage utilisé.

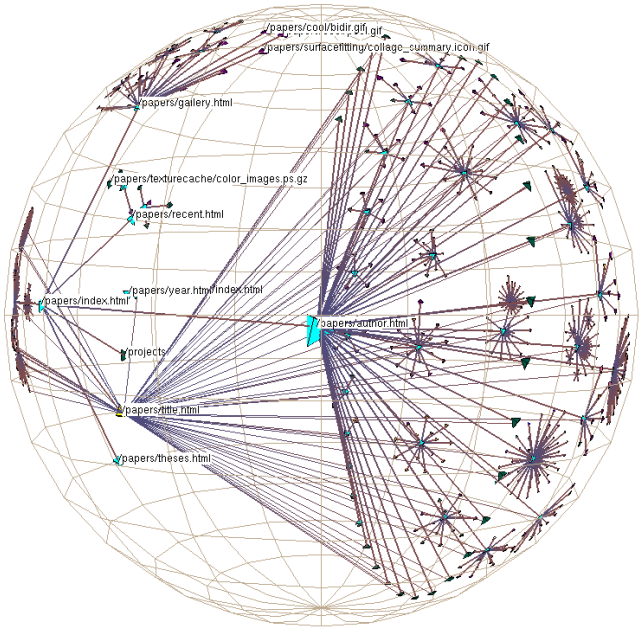


FIG. 2.12 – Graphe représentant un extrait des liens d'un site web dans un espace 3D hyperbolique. Extrait de [100].

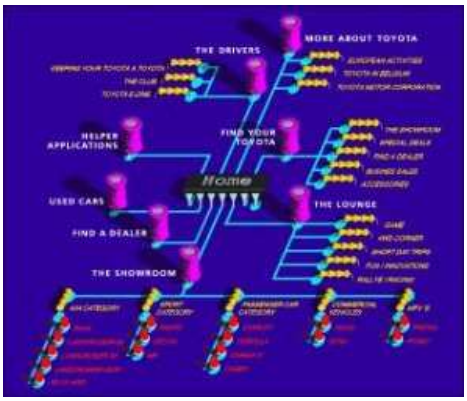


FIG. 2.13 – Utilisation de la métaphore du circuit électronique pour représenter un arbre.

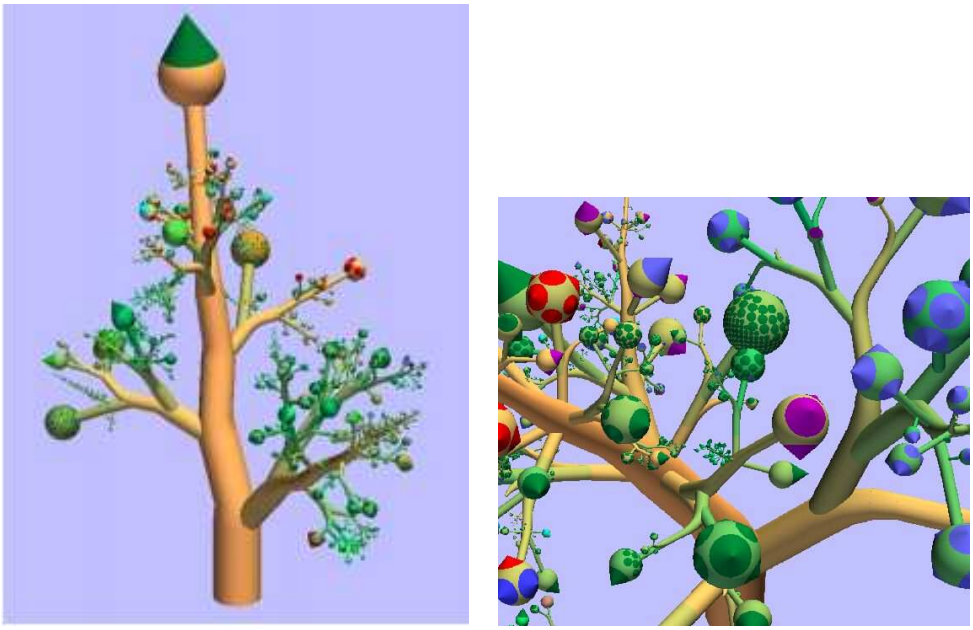


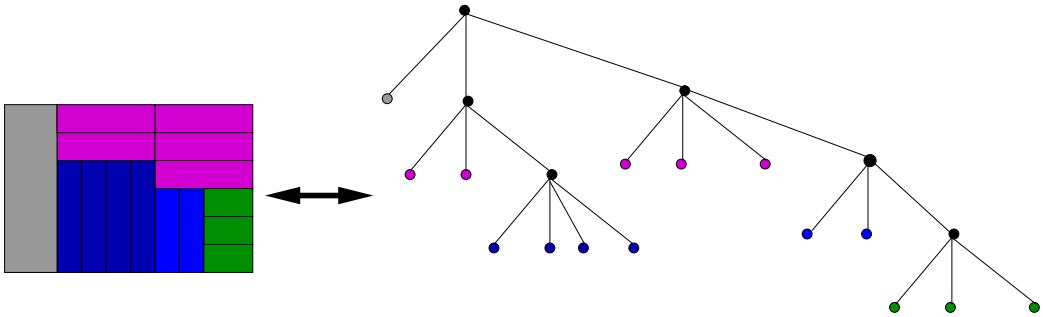
FIG. 2.14 – Visualisation d'un système de fichiers par une métaphore botanique [74].

2.6 Les langages de description

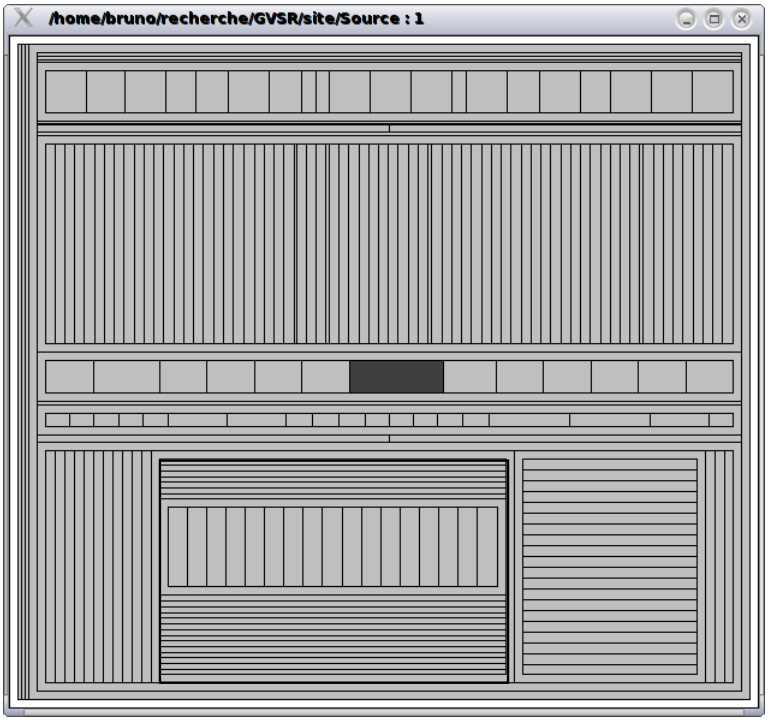
Chaque communauté d'utilisateurs utilise souvent ses propres logiciels de représentation plus ou moins adaptés spécifiquement aux besoins d'un domaine. Malheureusement, la grande majorité de ces logiciels utilise leur propre format de description de données bien souvent sous la forme d'un fichier texte. Par conséquent, la mise en place d'un jeu d'essai commun à plusieurs logiciels pour, par exemple comparer leurs fonctionnalités respectives, n'est pas évidente ; les possibilités d'importation ou d'exportation avec différents langages de description de données étant bien souvent limitées. Cependant, stimulées par l'émergence récente des solutions utilisant XML[8], des tentatives d'uniformisation sont en développement. D'une façon générale, les langages de description permettent pour la plupart de simplement définir un graphe en décrivant l'ensemble de ses sommets et de ses arêtes avec leurs attributs de style (*e.g.* couleurs, formes, tailles, noms). On peut notamment citer parmi les plus usités :

– **le langage dot** qui a été mis au point pour la suite logicielle *graphviz*

[8] *Extensible Markup Language*, `http://www.w3.org/XML/`



(a) Principe de création d'un treemap.



(b) Exemple avec un système de fichiers d'environ une douzaine de répertoires et 240 fichiers réalisé avec TreeMap Java Library (http ://treemap.sourceforge.net). Chaque rectangle comme celui avec un fond foncé représente un fichier. Les regroupements de ces rectangles (exemple du rectangle avec des bords épais au bas de la figure) représentent les différents répertoires.

FIG. 2.15 – Les Treemaps : principe de construction et exemple sur un arbre de grande taille.

de AT&T[9] [45] permet notamment de tracer des graphes en niveaux. Ce langage permet en plus de regrouper des objets ayant des attributs communs pour former des sous-graphes.

- **le langage GML** (*Graph Modeling Language*), qui a été développé pour la plate-forme *Graphlet*[10], dans un but d'uniformisation des différents langages de description existant à l'époque de sa création. Par rapport au langage dot, GML permet de décrire des informations spécifiques nécessaires pour la visualisation d'information. Des informations arbitraires (association de données externes à un sommet par exemple) peuvent être ajoutées dans les descriptions des composantes du graphe. Ainsi un fichier GML peut permettre d'émuler de nombreux langages de description.
- **Les langages basés sur XML** comme **GraphXML** [60] qui est conçu comme un format d'échange entre applications de tracés et visualisation de graphes ou même avec d'autres types d'applications. Il a aussi pour avantage d'être plus générique et plus simple à utiliser que GML en ce qui concerne les données externes. On peut aussi citer **XGMML**[11] qui est une transformation du langage GML en XML. Plus récemment, le langage **GXL** [141] (*Graph eXchange Language*) vise à créer un format d'échange entre applications qui soit le plus standard possible. Ce langage permet notamment d'émuler tous ceux cités précédemment.
- **Les librairies en C++** comme **LEDA** (*Library of Efficient Data types and Algorithms*) qui est constituée d'un ensemble de classes avec ses propres structures de données, son propre système de gestion de mémoire et beaucoup de méthodes et d'algorithmes spécifiques pour la description et la manipulation de graphes d'un point de vue plus théorique que les langages précédents [96]. On peut aussi citer la librairie **GTL** (*Graph Template Library*)[12], qui peut être vue comme une extension des librairies STL[13] pour la manipulation et la description des graphes.

[9] `http://www.graphviz.org/`

[10] `http://www.infosun.fmi.uni-passau.de/Graphlet/`

[11] Une version de travail des spécifications est disponible à : `http://www.cs.rpi.edu/~puninj/XGMML/draft-xgmml-20010628.html`

[12] `http://infosun.fmi.uni-passau.de/GTL/`

[13] *Standard Template Librairies*, `http://www.sgi.com/tech/stl/`

2.7 Conclusion

La représentation visuelle des graphes repose sur une procédure complexe qui intègre bien souvent des critères subjectifs liés à la perception du tracé par les utilisateurs. A cela, il faut ajouter les différentes contraintes engendrées par les données qui sont plus ou moins difficiles à satisfaire. Les différentes sections de ce chapitre montrent que :

1. pour le tracé statique, il est important de présenter à l'utilisateur un tracé lisible et compréhensible car ce dernier, s'il est souvent spécialiste des données, ne l'est pas des méthodes employées pour les représenter. Ces caractéristiques sont calculées différemment selon le type de graphe à tracer ;
2. l'extension de ces méthodes pour le tracé dynamique doit toujours satisfaire ces contraintes. Mais il faut en plus préserver au mieux la carte mentale de l'utilisateur entre deux étapes successives du tracé ;
3. pour le tracé des grands graphes, la satisfaction des contraintes passe par une adaptation des méthodes utilisées précédemment. Une conséquence immédiate due à la taille des structures utilisées est que l'utilisateur ne peut plus la visualiser intégralement et exhaustivement sur une représentation unique ;
4. pour le tracé en trois dimensions, dont la valeur ajoutée reste encore un sujet d'étude, les nouveaux problèmes posés (*e.g.* l'occlusion) rendent plus difficile la satisfaction des contraintes de lisibilité du tracé ;
5. d'autres approches s'affranchissent de la représentation conventionnelle "sommets-liens". Les plus abouties sont basées sur des codages orthographiques. Les métaphores graphiques, et leurs prolongements avec des représentations dans des espaces immersifs sont en plein développement.

La puissance actuelle des ordinateurs permet l'accès à des graphes de toutes tailles pouvant comporter jusqu'à plusieurs dizaines de milliers de sommets, la Toile fournissant des exemples extrêmes. Cependant, outre les difficultés techniques liées à la variabilité des environnements informatiques, l'absence d'un langage standard de description de graphes communément adopté conduit bien souvent l'utilisateur à se restreindre dans ses usages. Dans la lignée de l'essor des méta-langages en gestion des connaissances, les développements actuels (ex. GXL) permettront peut être de faciliter dans un futur proche l'interopérabilité entre les représentations et leurs comparaisons.

Les différentes méthodes présentées, ainsi que les langages de description de graphes associés peuvent, sûrement aider un utilisateur novice dans le

choix d'une méthode de représentation adaptée à sa problématique. Mais elles sont pour la plupart implémentées dans de nombreux logiciels dont le choix peut s'avérer difficile. Pour aider l'utilisateur dans ce choix, nous avons développé un site web, appelé GVSR (*Graph Visualization Software References*), présenté comme un annuaire, qui recense pour le moment une cinquantaine de logiciels présentés dans un format homogène sous la forme de fiches descriptives. Ce site est présenté dans l'annexe A page 143.

Chapitre 3

Tracé de graphes en niveaux

Sommaire

D'une façon générale, un tracé en niveaux, encore appelé tracé hiérarchique, d'un graphe consiste à affecter les sommets sur des niveaux verticaux ou horizontaux prédéfinis ; les arcs reliant des sommets de niveaux différents. Outre notre problématique en gestion des connaissances, cette convention de tracé est bien adaptée dans de nombreuses applications (représentation de structures de données, cartographie de réseaux, représentation d'organigrammes, ...)[1] pour mettre en évidence une structuration sur l'ensemble des sommets.

[1]De nombreux exemples sont disponibles en ligne. Voir notamment `http://www.`

Comme nous l'avons vu dans le chapitre précédent, un des critères majeurs retenus pour quantifier la lisibilité de ces tracés est celui de la minimisation des croisements d'arêtes. Ce problème peut sembler à priori plus simple dans le cas des graphes en niveaux que dans le problème général de minimisation du nombre de croisements d'arêtes dans un graphe quelconque dans le plan. Le nombre de croisements dépend ici de l'ordre des sommets dans les niveaux et non de leurs positions géométriques. Cependant le problème reste *NP*-difficile même s'il n'y a que deux niveaux [48].

Du fait de son importance applicative, depuis les travaux précurseurs de Carpano et Warfield [19, 138], de nombreuses approches ont été proposées [7]. Certaines sont basées sur une extension du problème à deux niveaux [32, 69] (se reporter à un article récent de Martí et Laguna [93] pour une comparaison de 14 méthodes[2]). D'autres travaux sur le problème multi-niveaux ont montré expérimentalement l'intérêt des métaheuristiques. Le choix d'une approche pouvant être délicat puisqu'il est subordonné à plusieurs critères : qualité des solutions, temps de calculs, ... Nous avons cherché, avant de rentrer dans la résolution, à mieux comprendre les spécificités de l'espace de recherche. Rosete-Suárez et *al.* [120, 121] ont souligné que la difficulté de ce problème est difficile à évaluer. Le critère classique qui utilise la taille du problème comme mesure de la complexité –ici l'ordre du graphe– n'est pas suffisant, la densité du graphe jouant aussi un rôle important. Ils mettent en avant la multimodalité de l'espace de recherche pour justifier la difficulté du problème. Cependant, à notre connaissance, cette supposition n'a pas été accompagnée de validations expérimentales significatives.

Afin de mieux comprendre les spécificités de l'espace de recherche, nous avons mené, en nous basant sur le modèle de Jones et Forrest [65] une analyse statistique dans deux directions. En nous restreignant tout d'abord à une famille de différents petits graphes et à des opérateurs locaux de transformation classiquement utilisés sur ce type de tracés (inversion de sommets, placement des sommets selon leur position médiane ou barycentrique), nous avons calculé exhaustivement les espaces de recherche associés. Cette première étude nous a permis de confirmer la présence de nombreux optima locaux et globaux et de préciser quelques propriétés de leurs bassins d'attraction. Puis, nous avons étendu notre analyse à des graphes de tailles plus élevées en recourant à une exploration *via* des descentes lancées en parallèles. Si les 5000 descentes que nous avons utilisées ne permettent évidemment pas de rendre compte de la complexité de l'espace de recherche, les résultats per-

`graphviz.org/Gallery.php` et `http://www.aisee.com/gallery/`

[2] La conclusion de l'article est que les métaheuristiques donnent de meilleurs résultats que les méthodes exactes plus classiques mais avec parfois un temps de calcul moins bon.

mettent de donner quelques arguments en faveur des méthodes stochastiques capables de changer de bassin d'attraction tardivement dans le processus de recherche.

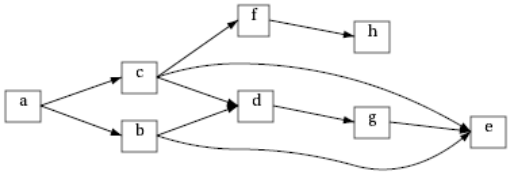
Ce chapitre commence par rappeler dans le paragraphe 3.1 une description formelle du problème de tracé en niveaux qui est utilisée dans le reste de la thèse. Le paragraphe 3.2 présente un état de l'art des différentes méthodes publiées pour le problème de tracé statique et les approches existantes pour le tracé dynamique qui connaît un regain d'intérêt. Le paragraphe 3.3 rappelle une description formelle des espaces de recherche dans le cas général et le paragraphe 3.4 présente notre modélisation des espaces de recherche pour le graphe en niveaux. Nous avons utilisé une modélisation originale sous la forme d'un graphe en niveaux. Les paragraphes 3.5 et 3.6 présentent les études statistiques de ces espaces de recherche.

3.1 Formalisation du problème de tracé

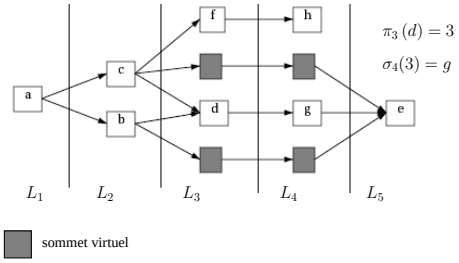
On considère dans la suite un graphe en niveaux acyclique $G = (V, E)$ avec V un ensemble de n sommets et E un ensemble de m arcs et une partition L de V en h niveaux $L_1, L_2, \ldots, L_h$. Les niveaux sont tels que si un arc $(u, v) \in E$, avec $u \in L_i$ et $v \in L_j$, alors $i < j$. La longueur de cet arc est $j - i$. Le nombre de sommets pour chaque niveau L_i est noté n_i. De plus, nous pouvons sans conséquence nous restreindre à des graphes propres qui ont la propriété d'avoir des arcs dont la longueur est fixé à 1. Cette propriété est obtenue en remplaçant un arc de longueur λ par un chemin composé de $\lambda - 1$ sommets virtuels sur chaque niveau intermédiaire (figure 3.1). Par convention, nous considérons que les différents niveaux sont représentés verticalement.

L'ordre des sommets sur chaque niveau L_k est défini par $\pi_k : L_k \rightarrow \{1, 2, \ldots, n_k\}$, où $\pi_k(u) = i$ signifie que le sommet $u \in L_k$ est à la position i sur L_k. Inversement, $\sigma_k(i) = \pi_k^{-1}(u)$ indique le sommet qui se trouve à la position i dans le niveau L_k. Un tracé de G est donc l'ensemble des ordres $\Pi = \{\pi_1, \pi_2, \ldots, \pi_h\}$ définis sur chaque niveau L_k.

Si l'on considère comme critère la minimisation des croisements d'arcs, le problème du tracé du graphe G se pose comme le problème d'optimisation suivant : trouver un ordre optimal des sommets $\widehat{\Pi}$ appartenant à l'ensemble des tracés possibles Ω du graphe G qui réduise au maximum le nombre de croisements d'arcs, noté $c(\Pi)$. Comme nous l'avons indiqué dans l'introduction de ce chapitre, ce problème est *NP*-difficile [48].



(a) Un tracé avec des arcs de longueurs variables.



(b) Le même tracé avec des sommets virtuels.

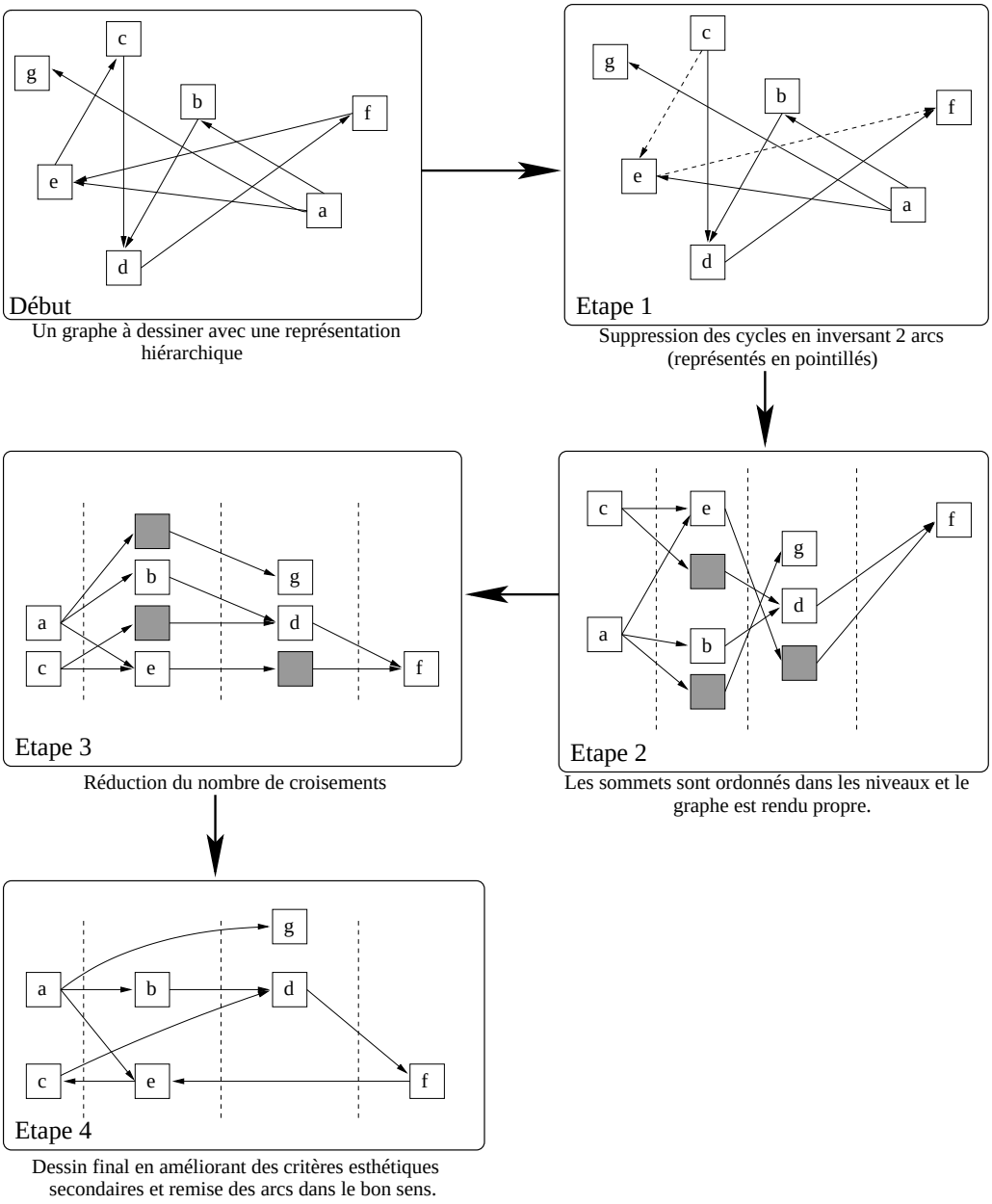
FIG. 3.1 – Transformation d'un graphe en un graphe propre avec des sommets virtuels.

3.2 Etat de l'art pour le problème statique

La majorité des méthodes de tracés pour le problème multi-niveaux se basent, au moins partiellement, sur l'heuristique de Sugiyama *et al.* [128] qui permet de dessiner des graphes en niveaux sans cycle en minimisant le nombre de croisements d'arcs. Cette heuristique est implémentée dans de nombreux logiciels de manipulation de graphes tel que le fameux *"dot"* de la suite *Graphviz* de AT&T [37]. La version de base se décompose en quatre étapes (voir la figure 3.2) :

1. **Suppression des cycles.** Obtenue, pour chaque cycle, en inversant un arc bien choisi[3]. A la fin de l'algorithme, l'arc retrouve son sens initial.
2. **Mise en place des sommets dans les niveaux.** Les sommets sont rangés dans les niveaux de telle sorte que tous les arcs aillent dans le même sens. Le graphe est ensuite rendu propre. Les sommets virtuels sont nécessaires pour la plupart des méthodes de réduction du nombre de croisements d'arcs. Ils sont supprimés lors de la dernière étape.

[3]La complexité de l'algorithme de détection des circuits dans un graphe est simplement fonction du nombre de sommets [97] alors que le problème de leur suppression est *NP*-difficile.



Soit un graphe G *avec* $V = \{a, b, c, d, e, f, g\}$ *et* $E = \{(a, b), (a, g), (a, e), (b, d), (c, d), (e, c), (d, f), (e, f)\}$.

FIG. 3.2 – Les différentes étapes de l'heuristique de Sugiyama pour le tracé de graphes niveaux [27, 34].

3. **Réduction du nombre de croisements.** Elle est obtenue par l'application de diverses méthodes d'optimisation, présentées ci-dessous.
4. **Affectation des coordonnées géométriques aux sommets.** A chaque niveau est associé une coordonnée sur l'axe des abscisses pour les sommets dans le cas d'une représentation verticale. L'ordonnée est calculée en optimisant des critères esthétiques secondaires tels que la minimisation de la longueur des arcs ou le respect du rapprochement de certains sommets à cause de leur sémantique associée.

Les méthodes de réduction des croisements peuvent se classer en deux grandes classes : les méthodes locales souvent déterministes, et plus récemment les métaheuristiques (recherche Tabou, GRASP et algorithmes génétiques).

3.2.1 Transformations locales

Ces méthodes ont été les premières à être publiées (voir Laguna *et al.* [83] pour une chronologie détaillée). Elles suivent pour la majorité d'entre elles le même principe : les sommets d'un niveau sont réordonnés pendant que l'ordre des sommets dans les autres niveaux reste fixe. Les différents niveaux du graphe sont parcourus séquentiellement les uns après les autres. Il existe deux grandes familles d'heuristiques : *(1)* celles qui se basent sur la permutation des sommets dans les niveaux en utilisant directement le nombre de croisements comme dans les tris classiques [138] ; *(2)* des heuristiques basées sur le calcul d'une moyenne[4] dont l'idée sous-jacente est que le nombre de croisements diminue si un sommet est placé à peu près au milieu de ses voisins sur les niveaux adjacents. Ceci revient à chercher les tracés où les arcs sont les plus horizontaux possibles. Le calcul des moyennes est basé sur celui d'un barycentre [128], d'une médiane [35], ou des variantes pouvant utiliser les deux opérateurs [47, 122].

3.2.2 Métaheuristiques

Recherche Tabou

Si les origines de cette métaheuristique remontent à la fin des années 70 son essor date des années 80 [50]. La spécificité de cette méthode est de diriger un processus de recherche en lui imposant des restrictions pour bien appréhender les "passages difficiles" et ainsi éviter des opérations inutiles (d'où le terme tabou qui est ici simplement synonyme d'interdit) [81].

[4] *Averaging heuristics*

Une application pour le tracé de graphes en niveaux a été publiée par Laguna *et al.* en 1997 [83] et adaptée ensuite pour le problème à deux niveaux [91]. Pour le problème multi-niveaux, l'algorithme combine deux étapes qui peuvent être répétées plusieurs fois : *(1)* la recherche d'un optimum local et *(2)* une recherche dans le voisinage de cet optimum pour essayer de l'améliorer.

La première étape consiste à sélectionner les niveaux selon leur importance à engendrer des croisements (l'importance d'un niveau est la somme du degré de ses sommets). Ensuite chaque sommet est permuté avec les autres sommets de son niveau. L'inversion qui est conservée est celle qui supprime le plus grand nombre de croisements. En cas d'égalité entre plusieurs solutions, celle qui place les sommets au plus près de leur barycentre est conservée. Dès qu'un niveau est entièrement réorganisé, il est marqué tabou et ne peut plus être sélectionné. Il cesse d'être tabou dès qu'un de ses niveaux adjacents est réorganisé. Cette première étape s'arrête quand tous les niveaux sont tabous.

Ensuite la phase de recherche locale consiste à choisir aléatoirement un sommet u et à le placer à $\pi(u)+1$ ou $\pi(u)$ ou encore $\pi(u)-1$ selon la position qui minimise le nombre de croisements. Cette étape de recherche locale est répétée $25 \times n$ fois, le nombre d'itérations étant fixé par les auteurs.

Les expérimentations ont été effectuées sur un ensemble de 180 graphes dont l'ordre de la majorité est compatible avec un tracé sur un écran d'ordinateur (les autres graphes ont un ordre très important). Deux versions de la recherche Tabou sont présentées : une qui fournit des résultats rapidement et l'autre qui fournit des résultats de la meilleure qualité possible. Chaque version donne des résultats meilleurs que les heuristiques basées sur les transformations locales avec une domination de la version centrée sur la qualité des solutions engendrées. Le temps de calcul de la version rapide est équivalent à celui des méthodes basées sur les transformations locales.

GRASP

GRASP pour *Greedy Randomized Adaptive Search Procedures* [42, 114] a été développé à la fin des années 80. C'est une heuristique à départs multiples facile à implémenter et qui ne nécessite que peu de paramètres. Elle se décompose en deux étapes. La première consiste à construire de façon judicieuse, élément par élément, une solution initiale du problème à traiter. La deuxième étape est une recherche locale dirigée pour améliorer significativement la solution construite précédemment. Un avantage important de GRASP est que si l'amélioration qui peut être obtenue risque d'être faible (calcul d'une probabilité), alors l'algorithme n'effectue pas les calculs.

Une version pour le problème du tracé à deux niveaux a d'abord été

publiée [82] puis elle a été étendue pour le problème multi-niveaux [92]. La construction de la solution initiale s'effectue sommet par sommet. Un premier sommet est choisi aléatoirement parmi ceux ayant un degré maximal pour le graphe à construire. Puis ce sommet est placé sur son niveau à une position aléatoire. Les sommets suivants sont positionnés selon le calcul d'un barycentre en utilisant les sommets déjà placés. Le choix des sommets à placer s'effectue par ordre décroissant de leur degré. En effet on peut intuitivement penser qu'un sommet avec un degré élevé induit plus de croisements d'arcs qu'un sommet avec un degré plus faible. La phase de recherche locale est ensuite appliquée sur chacun des sommets pris par ordre décroissant de leur degré. Dans la mesure du possible, cinq positions différentes sont testées de $\pi_k(u) - 2$ jusqu'à $\pi_k(u) + 2$. Le sommet est placé à la position qui minimise le nombre de croisements. La phase d'amélioration est répétée tant qu'au moins un sommet change de place et si l'algorithme estime que le nombre de croisements sera réduit de façon significative lors de l'étape suivante.

Les expérimentations effectuées sur un ensemble de 180 graphes, équivalent à celui utilisé pour la recherche Tabou, montrent que les tracés obtenus sont de meilleures qualités que ceux obtenus avec les transformations locales pour des temps de calculs équivalents. Les résultats sont toutefois moins bons que ceux obtenus avec la méthode Tabou.

Recuit simulé

Davidson et Harel [25] ont présenté une méthode basée sur du recuit simulé pour des graphes non orientés. La fonction de coût essaye de modéliser la qualité visuelle du tracé : les utilisateurs peuvent directement paramétrer des poids sur les divers critères esthétiques utilisés (nombre de croisements, distribution des sommets dans l'espace, longueur des arcs). La méthode du recuit-simulé est connue pour avoir des temps de calcul importants et ne peut donc être utilisée que pour des graphes dont l'ordre est faible.

3.2.3 Algorithmes génétiques

Les résultats encourageants des algorithmes génétiques dans de nombreux problèmes d'optimisation combinatoire, où le calcul d'une solution exacte est prohibitif ou tout simplement impossible, ont stimulé différents développements pour des problèmes de tracés. Suite au travaux précurseurs de Groves *et al.* [53] et Kosak *et al.* [78] de nombreuses méthodes souvent dérivées de ces deux premières sont apparues. L'atout majeur avancé des algorithmes génétiques par rapport aux autres méthodes pour le tracé de graphes est leur facilité à prendre en compte simultanément de nombreux critères esthétiques

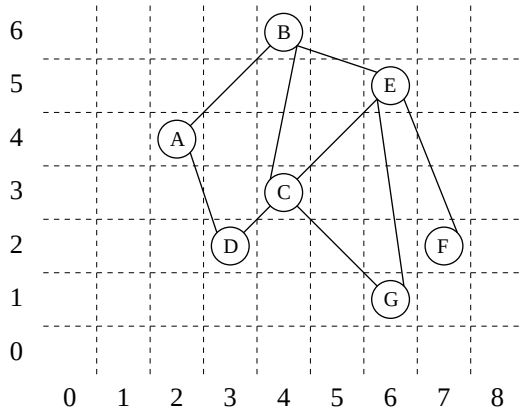


FIG. 3.3 – Exemple de graphe non orienté dont les sommets sont placés sur une grille.

qui peuvent être définis par l'utilisateur dès qu'un codage informatique est possible [119]. Ils sont aussi adaptables aux différents types de graphes tels que les graphes non orientés [13, 38, 61, 76, 79, 119, 121, 130] ou orientés [89, 94, 105, 132].

Les algorithmes sont bien souvent multiobjectifs et prennent en compte de nombreux critères esthétiques. On retrouve le plus souvent : réduction du nombre de croisements d'arêtes, réduction de la longueur des arêtes ou de la distance entre les sommets, réduction des angles entre des sommets adjacents, ... [13, 61, 79]. Un critère récurrent, suggéré par Hobbs et Rodgers [61], est d'utiliser au maximum l'espace alloué au tracé. Cet espace est découpé comme une grille et le problème se résume à optimiser les coordonnées des sommets sur la grille (figure 3.3) [38, 130]. Ces critères peuvent aussi être pondérés selon leurs importances pour l'utilisateur [121].

Les différentes expérimentations montrent que le choix de bons opérateurs de croisements, le plus souvent dédiés au problème, est fondamental. Ces opérateurs entraînent l'apparition de nombreuses contraintes supplémentaires qui augmentent sensiblement le temps de calcul. Ils sont souvent difficiles à concevoir [38]. Branke *et al.* [13] montrent même que parfois la combinaison de deux individus bien adaptés peut ne pas produire un individu encore mieux adapté mais au contraire un individu très mauvais (le principe des schémas ou briques de bases [51] ne s'applique pas). Ils suggèrent même de supprimer les opérateurs de croisements et de n'utiliser que des stratégies évolutionnaires basées exclusivement sur des opérateurs de mutations.

La majorité des auteurs concluent que les algorithmes génétique trouvent un bon tracé rapidement mais que le temps nécessaire pour affiner ce tracé et

détecter qu'il est de bonne qualité est très important. Deux approches sont envisagées pour résoudre ce problème. La première consiste à hybrider l'algorithme par une phase de recherche locale, basée sur une stratégie de descentes utilisant des transformations locales [61] ou des algorithmes spécifiques [13] (algorithmes à modèle de force). En utilisant des méthodes connues pour trouver de bon résultats rapidement, cette hybridation contribue à améliorer le temps de convergence des algorithmes [132]. La deuxième approche est l'utilisation de stratégies élitistes lors de la sélection des individus. Elle consiste à préserver le meilleur individu [13, 38, 130].

Pour le tracé orienté, une méthode originale publiée par Utech *et al.* [132] pour les graphes orientés sans circuit se base sur l'heuristique de Sugiyama. Elle permet de simultanément ranger les sommets dans les niveaux et optimiser le nombre de croisements. Même avec cette dernière méthode optimisée spécifiquement pour ce type de tracé, les temps de calcul restent élevés par rapport à ceux obtenus avec des transformations locales pour des qualités de résultats équivalentes.

3.2.4 Grimpeurs stochastiques

Le principe général d'un grimpeur stochastique est simple : une solution aléatoire du problème à traiter est créée puis elle est améliorée par des applications successives d'un opérateur de transformation tant que cela est possible. Bien souvent plusieurs grimpeurs sont lancés en parallèle.

Grâce à leur simplicité de mise en œuvre, les grimpeurs stochastiques peuvent être utilisés pour la comparaison des performances avec d'autres méthodes d'optimisation. En effet, Rosete *et al.* [120] regrettent qu'ils ne soient pas plus souvent utilisés dans ce cadre car leur simplicité permet bien souvent de trouver de bons résultats rapidement. En s'appuyant sur des expérimentations menées sur un problème de tracé de graphes simples, ils remarquent aussi que même si la meilleure solution n'est pas toujours trouvée par les grimpeurs stochastiques, ils terminent bien souvent sur une solution acceptable bien plus rapidement que d'autres méthodes basées sur des stratégies évolutionnaires.

3.2.5 Extension au tracé dynamique

Certains auteurs remarquent que leur méthode est inutilisable pour du tracé dynamique, bien souvent à cause du temps de calcul trop important. C'est le cas pour la version produisant les meilleurs résultats possibles pour la méthode basée sur la recherche Tabou [83], ou celle avec le recuit simulé [25].

Pour cette dernière méthode, les auteurs ont tout de même ajouté la possibilité d'animer les différentes étapes de l'optimisation du tracé. En revanche, d'autres auteurs ont intégré la contrainte du tracé dynamique pendant la conception de leurs algorithmes. D'une façon générale, toutes les méthodes conçues pour le tracé statique qui ont des temps de calcul courts, sont de bonnes candidates pour le tracé dynamique comme les transformations locales et GRASP. Certains algorithmes ont directement été conçus pour le tracé dynamique :

- *GALAPAGOS* de T. Masui [94] est un algorithme génétique qui permet de spécifier dynamiquement des contraintes sur le tracé du graphe. En revanche, il ne prend pas en compte l'ajout ou la suppression de sommets et/ou d'arcs ;
- certaines méthodes permettent directement à l'utilisateur d'agir sur le processus de tracé comme dans [80] ;
- ces actions de l'utilisateur peuvent se traduire par l'ajout de contraintes sur le tracé comme dans les travaux de Böhringer et Paulisch [17], He et Marriott [57], ou le système *GDHints* de Nascimento et Eades [29]. Ce dernier est basé sur un algorithme génétique dynamique, qui est lui même une extension de leur travail sur l'heuristique de Sugiyama à laquelle est ajoutée une gestion de contraintes [30]. Dans ces différents travaux, le côté dynamique ne signifie pas que l'utilisateur regarde simplement le graphe évoluer. Il participe aussi activement au processus en guidant l'algorithme vers des parties du tracé non optimisées ou en ajoutant des contraintes sur le placement des sommets afin de réduire l'espace de recherche ;
- *Dynadag* de North *et al.* [104, 37] est une version spécifique de l'algorithme *dot* qui est lui même basé sur l'heuristique de Sugiyama.

3.3 Les paysages de recherche

Avant de concevoir un algorithme pour la résolution d'un problème d'optimisation, il est utile de connaître au préalable les propriétés structurelles de l'espace de recherche[5].

Introduite par le biologiste S. Wright en 1932 pour illustrer sa théorie sur l'évolution des espèces [142], l'étude des paysages de recherche a été reprise en optimisation. Elle permet d'aider à la compréhension de la distribution des différentes solutions, et plus particulièrement leur qualité et le nombre d'optima locaux et globaux du problème. Selon Jones [65, 67], repris ensuite

[5] *fitness landscape*

par Vassilev [133], un paysage de recherche peut être défini par 3 composantes : le codage des solutions (génotype), une application $f : S \rightarrow \mathbb{R}$, où S est l'ensemble des solutions du problème traité, qui associe à chaque solution $s \in S$ une valeur numérique $f(s)$ (fonction objectif ou d'adaptation) et un opérateur de transformation ϕ qui définit une relation de voisinage sur l'ensemble des solutions. On suppose ici que S est fini.

3.3.1 Structure d'un paysage de recherche

Notion de voisinage

Cette notion est définie de façon univoque pour chaque opérateur de transformation. Soit ϕ un opérateur permettant de passer d'un point v du paysage à un autre point w et S_ϕ l'ensemble des points du paysage associé à ϕ. Le *voisinage* d'un point v pour l'opérateur ϕ est défini par :

$$N_\phi(v) = \{w \in S_\phi \,|\phi(v) = w\} \tag{3.1}$$

On dit alors que *w est voisin de v pour l'opérateur ϕ.*

Représentation des optima

On suppose ici que la fonction objectif f est une fonction d'adaptation à maximiser. Un optimum est alors un point du paysage dont tous les voisins ont une valeur d'adaptation inférieure. Un optimum est défini de façon univoque pour chaque opérateur ϕ. Soit $v, w \in S_\phi$, on dit que v est un *maximum pour son voisinage* si :

$$\forall w \in N_\phi(v) \Rightarrow f(v) \geq f(w) \tag{3.2}$$

v est aussi appelé un *optimum local.*

Un *optimum global* o est tel qu'il n'existe pas d'autre point du paysage qui ait une valeur de la fonction d'adaptation supérieure :

$$\forall w \in S_\phi \Rightarrow f(o) \geq f(w) \tag{3.3}$$

Plateaux

Un plateau est un ensemble de sommets M voisins au sens de ϕ qui ont tous la même valeur d'adaptation. Un plateau P peut se définir formellement comme :

$$\begin{gathered} M \subseteq S_\phi, |M| > 1 : \forall (v_i, v_j) \in M^2, f(v_i) = f(v_j) \\ \text{avec } \forall v_k \in M, \exists v_l \in M \,|v_k \in N_\phi(v_l) \end{gathered} \tag{3.4}$$

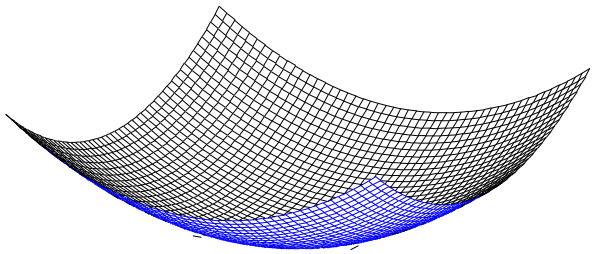


FIG. 3.4 – Surface quadratique

Bassin d'attraction

Un bassin d'attraction B_ϕ est un ensemble de points voisins au sens de ϕ qui permettent d'atteindre un optimum v_n par des transformations successives avec un accroissement strict de la fonction d'adaptation :

$$B_\phi(v_n) = \{v_0 \in S_\phi \,|\exists v_1, \ldots, v_n \in S_\phi \text{ avec } v_{i+1} \in N_\phi(v_i) \text{ pour } 0 \leq i \leq n\} \tag{3.5}$$

La taille d'un bassin d'attraction est le nombre de points du paysage qu'il contient.

3.3.2 Différents types de paysages

Lorsque l'on est confronté à un problème d'optimisation combinatoire, on ne connaît bien souvent pas *a priori* la structure du paysage de recherche. Cependant, leurs propriétés sont fondamentales pour le choix d'une méthode appropriée. Nous illustrons cette relation entre la structure de l'espace de recherche et le choix d'une méthode dans le cas continu en reprenant une petite typologie proposée par Renders [113] :

1. Les surfaces linéaires ou quadratiques (figure 3.4).
   Il existe des méthodes capables de résoudre de tels problèmes en un nombre fini d'itérations.
2. Les surfaces unimodales (figure 3.5).
   Plusieurs méthodes simples sont possibles tels que des grimpeurs stochastiques.
3. Les surfaces multimodales simples (figure 3.6).
   Des grimpeurs stochastiques travaillant en parallèle permettent d'obtenir des bons résultats.
4. Les surfaces "unimodale à gros grains" (figure 3.7).
   Ces surfaces peuvent être considérées comme des surfaces unimodales

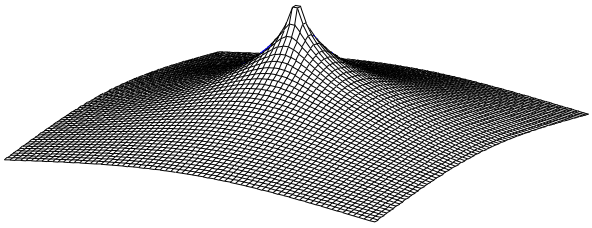


FIG. 3.5 – Surface unimodale

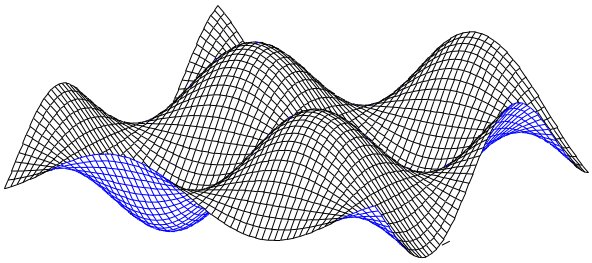


FIG. 3.6 – Surface multimodale simple

bruitées. Une méthode tolérant une dégradation modérée et temporaire des solutions pendant la recherche telle que le recuit-simulé [73] fournit de bons résultats.

5. Les surfaces multimodales à "structure combinative" (figure 3.8).
   Ici la position des optima locaux renseigne sur celle des optima globaux. Il faut utiliser une méthode capable d'exploiter la structure de la figure comme un algorithme génétique hybridé par un grimpeur stochastique. L'hybridation par une méthode de type grimpeur permet d'accélérer la recherche des optima locaux.
6. Les surfaces multimodales sans structure particulière (figure 3.9).
   Ces surfaces n'ayant aucune structure particulière, aucune méthode n'est vraiment adaptée. Une bonne solution est de recourir à des tirages

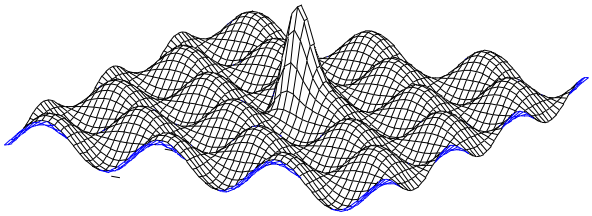


FIG. 3.7 – Surface "unimodale à gros grains"

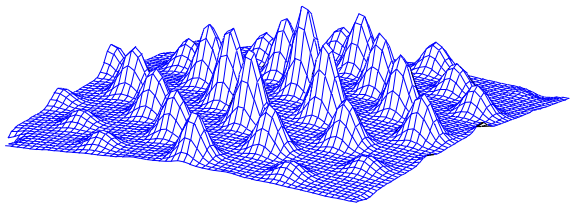


FIG. 3.8 – Surface multimodale à "structure combinative"

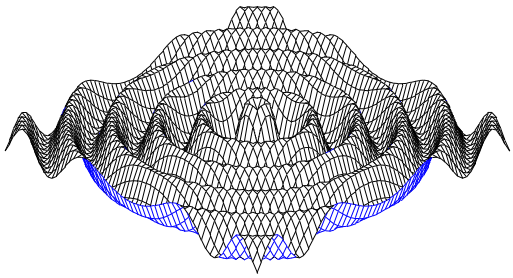


FIG. 3.9 – Surface multimodale sans structure particulière

aléatoires pendant un temps donné.

7. "L'aiguille dans la botte de foin" (figure 3.10).
   Sans commentaire.

8. Les surfaces avec des plateaux (figure 3.11).
   Ces surfaces sont difficilement exploitables par des grimpeurs stochastiques. En revanche les algorithmes génétiques sont efficaces grâce à leur capacité d'exploration de l'espace de recherche.

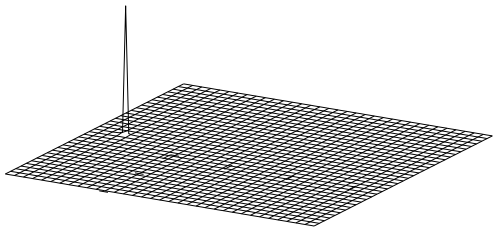


FIG. 3.10 – L'aiguille dans la botte de foin

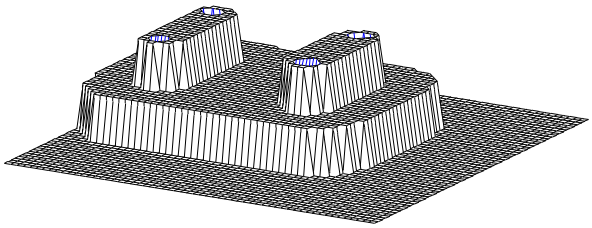


FIG. 3.11 – Surface avec des plateaux

3.4 Paysages de recherche pour le tracé de graphes en niveaux

Nous avons cherché à appliquer les concepts présentés précédemment au problème de tracé de graphes en niveaux. Dans ce cas, un paysage de recherche peut se modéliser par un graphe, appelé dans la suite "graphe-paysage" (GP). Les sommets D du GP sont les différents tracés possibles Ω du graphe G à tracer. Deux sommets D_1 et D_2 sont dans le même niveau s'ils ont la même valeur de la fonction d'adaptation, qui est simplement le nombre de croisements : $c(D_1) = c(D_2)$. Il existe un arc entre deux sommets D_0 et D_1 si le résultat de l'application d'un opérateur de transformation, noté O, sur le tracé représenté par D_0 permet d'obtenir le tracé représenté par D_1 : $D_1 = O(D_0)$. L'opérateur de transformation retenu consiste simplement à permuter deux sommets adjacents dans un niveau. Une application de l'opérateur n'est conservée que si le nombre de croisements est amélioré : $c(D_1) < c(D_0)$. La figure 3.12 est un extrait d'un GP associé à un petit graphe de 8 sommets.

Un *optimum local* est un sommet $\widetilde{D} \in \Omega$ tel que

$$c\left(\widetilde{D}\right) \leq c(D_N), \forall D_N \in N_O\left(\widetilde{D}\right) \tag{3.6}$$

où $N_O\left(\widetilde{D}\right)$ est l'ensemble des voisins de D au sens de l'opérateur O. Les sommets D_4 et D_5 sur la figure 3.12 sont des optima locaux.

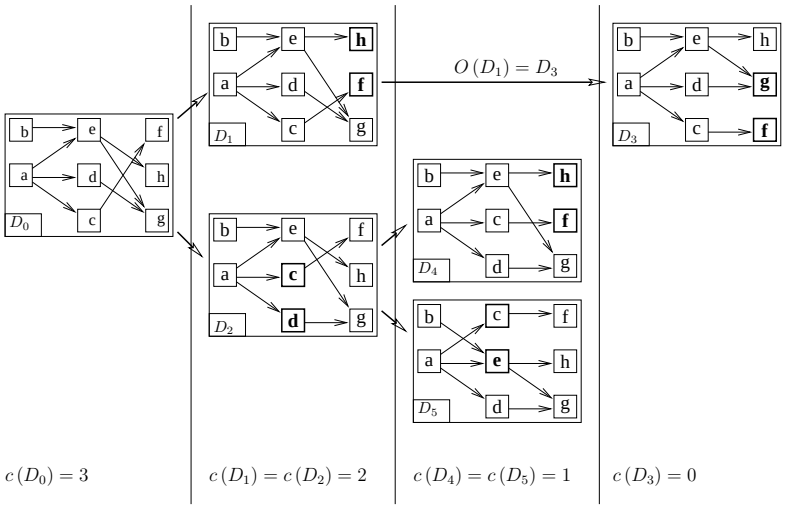
Un *optimum global* est un sommet $\widehat{D} \in \Omega$ tel que

$$c\left(\widehat{D}\right) \leq c(D), \forall D \in \Omega \tag{3.7}$$

Le sommet D_3 sur la figure exemple est un optimum global.

Le *bassin d'attraction* $B_O\left(\widehat{D_n}\right)$ d'un optimum $\widehat{D_n}$ est l'ensemble des sommets :

$$\begin{array}{c} B_O(D_n) = \{D_0 \in \Omega; \exists D_1, \ldots, D_{n-1} \in \Omega \\ \text{avec } D_{j+1} \in N_O(D_j) \text{ et } c(D_{j+1}) < c(D_j), \forall j = 0, \ldots, n-1\} \end{array} \tag{3.8}$$



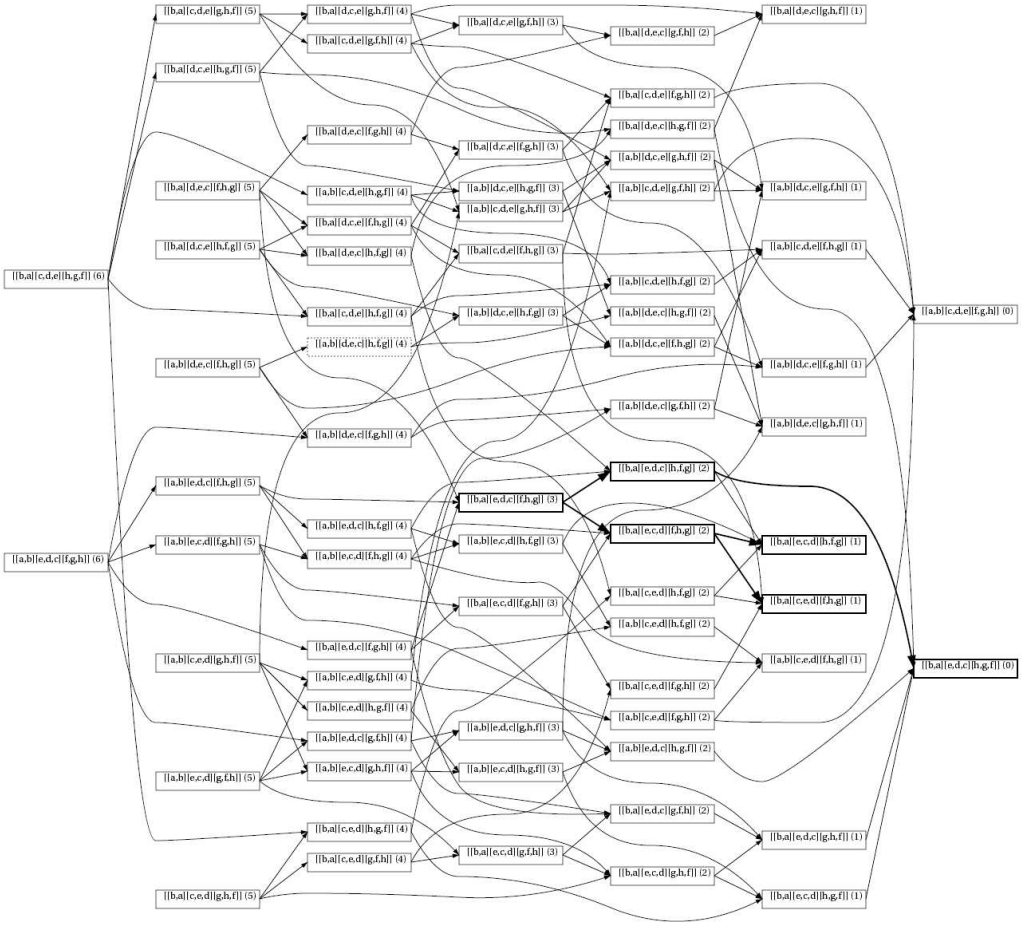
- *Les sommets inversés par une application de O sont dessinés en gras.*
- $B_O(D_3) = \{D_0, D_1\}$
- *D_3 est un optimum global. D_4 et D_5 sont des optima locaux.*

FIG. 3.12 – Extrait d'un graphe-paysage associé à un petit graphe.

Le bassin d'attraction de l'optimum global D_3 est composé de $B = \{D_0, D_1\}$. Le tableau 3.1 récapitule les différentes notions associées à un paysage de recherche avec leur équivalent dans un GP.

La figure 3.13 représente le GP complet associé à la figure 3.12. Les sommets les plus à gauche (resp. à droite) sont ceux ayant le plus (resp. le moins) de croisements. Une conséquence de la définition choisie de l'opérateur de transformation, qui ne retient que les applications améliorant le nombre de croisements, est qu'un graphe-paysage n'est pas forcément connexe. Une application de l'opérateur peut permettre de transformer un sommet d'un niveau en un autre sommet du même niveau. Nous avons choisi de ne pas représenter de tels arcs à cause de la complexité des calculs mis en œuvre par la suite et aussi dans un but de simplification des différentes représentations.

Notons que la définition 3.8 d'un bassin d'attraction pour les GP qui dérive du cas continu, permet des intersections non vides entre les bassins d'attraction des optima locaux et ceux des optima globaux. Sur la figure 3.14 qui est un zoom sur une partie du GP de la figure 3.13, le sommet représenté par un rectangle en pointillé (situé dans le niveau le plus à gauche) appartient à la fois au bassin d'attraction d'un optimum local et à celui d'un optimum global. Cette caractéristique souvent ignorée dans la littérature a des conséquences importantes sur les processus de recherche. De plus cet exemple confirme que le fait de ne conserver que le meilleur tracé après application

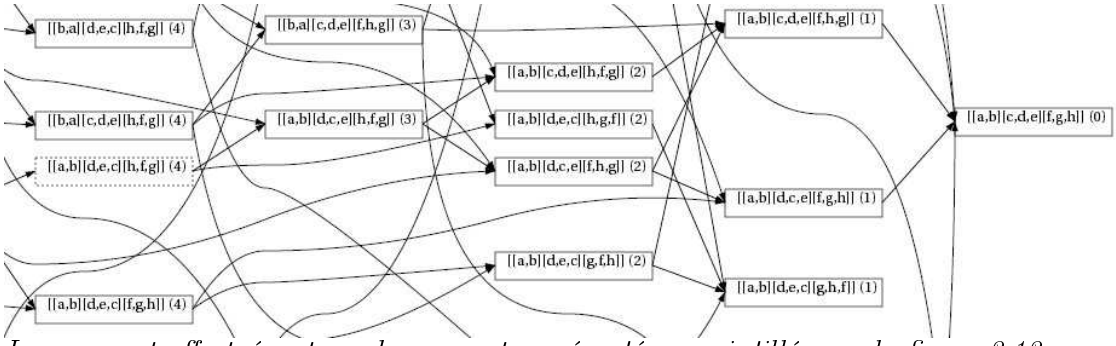


La portion de ce GP représentée avec des sommets et des arcs en gras est la figure 3.12.

FIG. 3.13 – Représentation d'un graphe-paysage pour l'opérateur d'inversion de sommets adjacents.

Notion	Paysage de recherche	Graphe-paysage
Points du paysage	x, v, o	D_n
Fonction d'adaptation	$f(x)$	$c(D_n)$
Opérateur de transformation	$\phi(x)$	$O(D_n)$
Voisinage	$N_\phi(x)$	$N_O(D_n)$
Optimum local	v	$\widetilde{D}$
Optimum global	o	$\widehat{D}$
Bassin d'attraction	$B_\phi(x)$	$B_O(D_n)$

TAB. 3.1 – La modélisation des paysages de recherche par un "graphe-paysage".



Le zoom est effectué autour du sommet représenté en pointillés sur la figure 3.13 (situé ici dans le niveau le plus à gauche). Il se situe dans le troisième niveau en partant de la gauche et c'est le huitième sommet en partant du haut.

FIG. 3.14 – Zoom sur le graphe-paysage de la figure 3.13.

de O, "descente" simple souvent utilisée en optimisation combinatoire, ne permet pas forcément de conduire à un optimum global.

L'ordre d'un GP pour l'opérateur O d'inversion de sommets adjacents est :

$$\prod_{k=1}^{h} n_k! \tag{3.9}$$

A titre d'illustration, malgré le faible nombre de sommets du graphe servant de support dans l'exemple, seulement huit sommets, le GP possède déjà une certaine complexité : il y a 72 sommets répartis en 2 optima globaux, 6 optima locaux, 14 sommets appartenant uniquement au bassin d'attraction d'un optimum global, 13 sommets appartenant uniquement au bassin d'attraction d'un optimum local et les 37 derniers sommets sont dans les deux types de bassins d'attraction.

3.5 Analyse statistique pour des petits graphes

Pour mieux se rendre compte de la difficulté du problème et essayer de construire une méthode d'optimisation adaptée, nous avons effectué une analyse statistique des GP associés à l'opérateur d'inversion de sommets adjacents pour un ensemble de petits graphes. Nous avons, pour cela, engendrés exhaustivement les GP associés à un ensemble Δ de 1875 petits graphes en niveaux qui ne soient pas des arbres, qui soient connexes et tels que $\prod_{k=1}^{h} n_k! \leq 2000$. L'ordre moyen des GP obtenus est de 925 sommets.

Description des GP

La première partie de l'analyse porte sur la distribution des optima et le comportement de la fonction d'adaptation. Ces informations renseignent directement sur la complexité du paysage.

Différentes approches statistiques ont été proposées pour analyser la distribution des optima. La *"fitness distance correlation"* (FDC) proposée par Jones et Forrest [66] est une mesure relativement populaire pour des problèmes dont les optima globaux sont connus. Elle permet de mesurer la corrélation entre différentes valeurs de la fonction d'adaptation avec la distance par rapport à un optimum global. Dans notre cas, plus on s'approche d'un optimum, plus la valeur de la fonction d'adaptation (le nombre de croisements d'arcs) est petite et donc plus la valeur de la FDC devrait tendre vers 1. Son calcul dépend du choix d'une métrique permettant de définir une distance

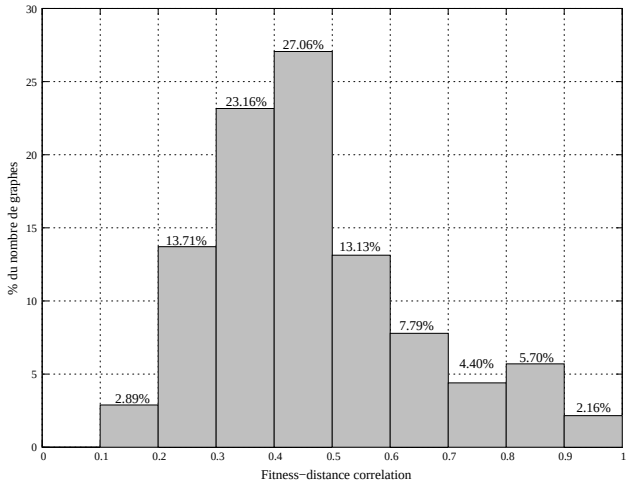


FIG. 3.15 – Distribution des valeurs pour la "*fitness-distance correlation*".

entre les sommets du paysage. Nous avons choisi une métrique simple basée sur le nombre de permutations entre les sommets. Soit D_i et D_j deux tracés appartenant à Ω avec σ_{ki} et σ_{kj} leur position respective sur chaque niveau l_k appartenant à l'ensemble des niveaux L_{GP} du GP. La distance entre D_i et D_j est définie par :

$$d\left(D_i, D_j\right) = \sum_{k=1}^{h} \left|C\left(\sigma_{ki}, \sigma_{kj}\right)\right| \text{ où } C\left(\sigma_{ki}, \sigma_{kj}\right) = \left\{u; \sigma_{ki}(u) \neq \sigma_{kj}(u)\right\} \quad (3.10)$$

On considère l'ensemble $C = \{c_1, c_2, \ldots, c_s\}$ des valeurs prises par la fonction d'adaptation pour un ensemble de s sommets et l'ensemble associé $D = \{d_1, d_2, \ldots, d_s\}$ des s distances de ces individus par rapport à l'optimum global le plus proche. La FDC est le cœfficient de corrélation r tel que :

$$r = \frac{cov_{CD}}{\sigma_C \times \sigma_D} \quad (3.11)$$

avec

$$cov_{CD} = \frac{1}{n} \sum_{i=1}^{h} \left(c_i - \overline{c}\right)\left(d_i - \overline{d}\right) \quad (3.12)$$

qui est la covariance de C et D, et σ_C, σ_D, $\overline{c}$ et $\overline{d}$ sont respectivement les écart-types et les moyennes des ensembles C et D.

La valeur moyenne de FDC sur l'ensemble des GP de Δ est 0.47. Cette valeur élevée est partiellement expliquée par la taille réduite des instances. Néanmoins, malgré cette petite taille, des variations importantes peuvent

apparaître (figure 3.15). L'échelle donnée par Jones et Forrest, indique que certaines instances peuvent être considérées comme faciles ou très faciles (FDC élevée) alors que d'autres sont difficiles (FDC faible) et nécessitent un nombre important d'étapes intermédiaires avant la convergence vers un optimum global.

Le calcul exhaustif de ces GP permet d'affiner ce premier résultat macroscopique par une analyse de la distribution des optima locaux et globaux. Il existe au moins un optimum local dans 76% des cas et la majorité des paysages sont multimodaux : le nombre moyen d'optima locaux est de 34.9 et celui des optima globaux de 36.1. La distribution de ces optima est très variable selon les graphes : l'écart-type de la moyenne du nombre d'optima locaux (resp. optima globaux) est de 37.6 (resp. 79.8).

Lorsque les optima locaux sont nombreux, il est important de connaître leur qualité, c'est à dire leur hauteur relative par rapport aux optima globaux. La hauteur relative $h(D_i)$ d'un optimum local D_i par rapport à un optimum global $\widehat{D}$ peut être mesurée en calculant le ratio suivant :

$$h(D_i) = 1 - \frac{c(D_i) - c\left(\widehat{D}\right)}{c(D_w) - c\left(\widehat{D}\right)} \tag{3.13}$$

où $c(D_w)$ est le nombre de croisements du plus mauvais dessin possible[6]. Si $h(D_i)$ est proche de 1, alors l'optimum local D_i peut être considéré comme une solution acceptable. La distribution de la hauteur des optima locaux pour Δ est donnée sur la figure 3.16. Il faut noter que 63% des optima ont une hauteur inférieure à 0.8 et ne peuvent donc pas être considérés comme des solutions acceptables.

Convergence sur un optimum local

Nous avons vérifié expérimentalement que les optima locaux sont nombreux et que leur qualité peut parfois être mauvaise par rapport à celle d'un optimum global. De plus, l'intersection des bassins d'attraction des optima locaux et globaux étant non vide, il est intéressant de connaître la propension d'une méthode simple à converger vers des optima locaux plutôt que vers des optima globaux. Nous avons estimé cette probabilité sur notre échantillon Δ pour une méthode de descente donnée dans l'algorithme 3.1.

Soit n_{no} le nombre de parcours effectués par la descente (par définition, c'est le nombre de sommets qui ne sont pas des optima) et n_l le nombre

[6] $c(D_w)$ est introduit ici car bien souvent les optima globaux des graphes de petites tailles sont sans croisement et aussi car nous avons généré les graphes-paysages exhaustivement.

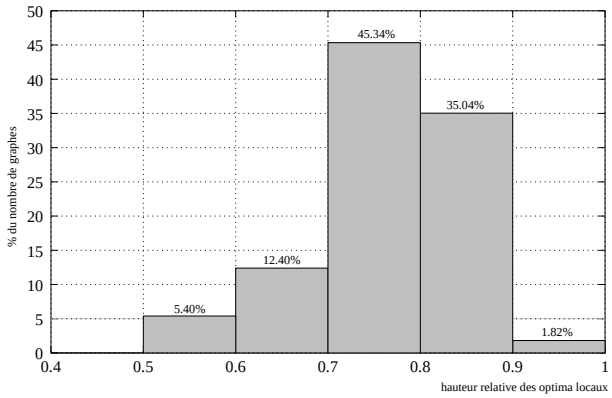


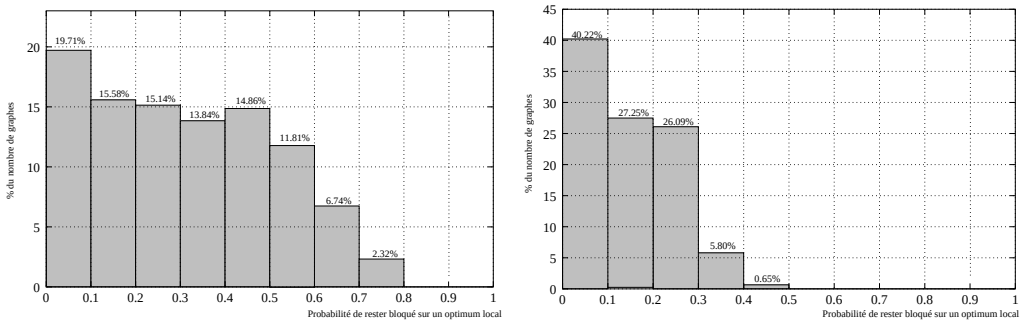
FIG. 3.16 – Hauteur relative des optima locaux.

Algorithme 3.1 Méthode de descente pour estimer la probabilité de trouver un optimum local vs global

Entrée : un GP à parcourir
Sortie : un optimum de ce GP
Variables : le nombre total de parcours n_{no}, n_l le nombre de parcours convergeant sur un optimum local et u, o des sommets du GP

Début
$n_{no} \leftarrow 0$
$n_l \leftarrow 0$
Pour chaque sommet u du GP **faire**
  Si u n'est pas un optimum **alors**
    $n_{no} \leftarrow n_{no} + 1$
    Répéter
      Choisir le meilleur voisin de u
    Jusqu'à trouver un optimum o
    Si o est un optimum local **alors**
      $n_l \leftarrow n_l + 1$
    Fin Si
  Fin Si
Fin Pour
Renvoyer n_{no} et n_l
Fin

Si pour le choix du meilleur voisin, plusieurs sommets ont le même nombre de croisements, un des sommets est choisi au hasard.



(a) Version 1 de l'heuristique de descente. (b) Version 2 de l'heuristiques de descente.

FIG. 3.17 – Probabilité d'atteindre un optimum local.

de parcours convergeant sur un optimum local. La probabilité p_l de rester bloqué sur un optimum local est donnée par n_l/n_{no}. L'estimation moyenne sur Δ est de 0.31. Bien que les calculs soient effectués sur des petits graphes et que tous les voisins de chaque sommet des GP soient connus, cette valeur est loin d'être négligeable.

Une fois encore, la distribution des valeurs n'est pas homogène (figure 3.17-a) : 19.71% des cas où $p_l < 0.1$ peuvent être considérés comme faciles, et 20.87% des cas où $p_l > 0.5$ peuvent être considérés nettement plus difficiles. Dans ces derniers, il est nécessaire de mettre en œuvre des stratégies de recherche plus adaptées aux spécificités du problème.

De par la forme particulière des bassins d'attraction, deux situations différentes sont possibles selon le sommet de départ de la descente : (i) le sommet appartient seulement au bassin d'attraction d'un optimum local ou global et (ii) ce sommet se situe à l'intersection des deux types de bassin d'attraction. Pour le deuxième cas, le choix du sommet suivant dans la descente est important. En effet comme nous l'avons déjà illustré précédemment, le fait de choisir le meilleur voisin peut ne pas faire converger la descente sur un optimum global. Pour mesurer l'importance de ce dernier point, nous avons légèrement modifié l'algorithme de la descente afin que le sommet de départ soit uniquement à l'intersection des deux types de bassins d'attraction. La nouvelle estimation moyenne de trouver un optimum local est de 0.14. Elle dépasse 0.2 pour 32% des graphes (voir figure 3.17-b pour la distribution des valeurs).

Cette valeur non nulle montre bien la complexité de l'espace de recherche pour le problème du tracé de graphes en niveaux. Les procédures appliquées classiquement en optimisation combinatoire utilisées ici (ex. choisir le meilleur voisin) ne sont pas suffisantes. Lors de l'optimisation d'un tracé,

des critères supplémentaires pour le choix d'un voisin ou des opérateurs de transformations plus complexes sont nécessaires.

3.6 Analyse statistique pour des grands graphes

Nous avons cherché à vérifier si les caractéristiques associées à la complexité de la recherche d'un optimum global se retrouvent sur des graphes plus grands dont les caractéristiques sont plus proches de celles des graphes que l'on peut rencontrer en Gestion des Connaissances. Cette étude approfondit un travail précédent mené par R. Lehn et P. Kuntz [87] sur une analyse partielle des graphes-paysages associés à des grands graphes avec un ensemble de 1000 descentes à départs multiples.

Soit Δ' un ensemble de 300 graphes en niveaux qui ne soient pas des arbres, qui soient connexes avec un nombre de niveaux compris entre 4 et 40 et un nombre de sommets par niveaux compris entre 3 et 15. La taille des GP associée est telle que $10^6 \leq \prod_{k=1}^{h} n_k! \leq 10^{14}$. Pour chaque graphe, un ensemble de 5000 tracés aléatoires est engendré et chaque tracé est amélioré par plusieurs applications de trois opérateurs O_1, O_2 et O_3 : ils sont successivement appliqués sur chaque niveau en suivant un principe de balayage de gauche à droite. Le processus commence par l'application de O_1, si le nombre de croisements diminue alors O_2 est appliqué et ainsi de suite jusqu'à ce que le nombre de croisements se stabilise ou soit nul (voir l'algorithme 3.2).

L'objectif ici n'est pas de développer une nouvelle heuristique mais de mieux connaître le paysage de recherche associé à des opérateurs qui soient proches de ceux de la littérature. Les transformations locales présentées au paragraphe 3.2.1 sont connues pour calculer des solutions dans des temps acceptables. Bien que des approches plus complexes ayant leur origine dans la conception des circuits intégrés aient été utilisées [41, 95], nous nous restreignons aux opérateurs les plus utilisés.

3.6.1 Opérateurs de transformations locales

Nous avons utilisé ici l'opérateur d'inversion de sommets adjacents déjà présenté (O_1), un opérateur basé sur le principe de calcul d'une médiane (O_2) inspiré de la définition de Sugiyama *et al.* [128] et enfin un opérateur basé sur un calcul de barycentre (O_3). Les définitions des opérateurs de médiane et barycentre données ci-dessous considèrent que la nouvelle position d'un sommet $u \in L_k$ après application des opérateurs dépend de ses voisins sur L_{k-1} et L_{k+1}. Cette définition est connue pour être moins performante dans

Algorithme 3.2 Procédure de descente pour l'analyse des GP des grands graphes.

Entrée : Un graphe G à tracer.
Sortie : Un tracé de G amélioré.

Début
$recommence \leftarrow 1$
Engendrer un tracé aléatoire Π de G
Tant Que $recommence == 1$ **faire**
  $recommence \leftarrow 0$
  Pour chaque niveau L_k de Π **faire**
    $\Pi_{O_1} \leftarrow \Pi$
    Appliquer O_1 sur L_k pour créer L'_k
    Remplacer L_k par L'_k dans Π_{O_1}
    Si $c(\Pi_{O_1}) < c(\Pi)$ **alors**
      $recommence \leftarrow 1$
      Afficher Π_{O_1} et $c(\Pi_{O_1})$
      $\Pi \leftarrow \Pi_{O_1}$
    Fin Si
    $\Pi_{O_2} \leftarrow \Pi$
    Appliquer O_2 sur L_k pour créer L'_k
    Remplacer L_k par L'_k dans Π_{O_2}
    Si $c(\Pi_{O_2}) < c(\Pi)$ **alors**
      $recommence \leftarrow 1$
      Afficher Π_{O_2} et $c(\Pi_{O_2})$
      $\Pi \leftarrow \Pi_{O_2}$
    Fin Si
    $\Pi_{O_3} \leftarrow \Pi$
    Appliquer O_3 sur L_k pour créer L'_k
    Remplacer L_k par L'_k dans Π_{O_3}
    Si $c(\Pi_{O_3}) < c(\Pi)$ **alors**
      $recommence \leftarrow 1$
      Afficher Π_{O_3} et $c(\Pi_{O_3})$
      $\Pi \leftarrow \Pi_{O_3}$
    Fin Si
  Fin Pour
Fin Tant que
Fin

l'heuristique de Sugiyama quand les sommets dont la position n'a pas encore été optimisée sont pris en compte dans les calculs. Mais comme ici l'opérateur est utilisé seul, la situation diffère. L'amélioration du nombre de croisements obtenue est probablement la combinaison de deux effets. Il a été montré que pour certains graphes l'opérateur de barycentre est mieux adapté que la médiane et vice-versa [27]. De plus, il peut y avoir un effet de bord dû à la définition choisie de la médiane qui ne prend pas en compte explicitement la parité.

O_2 : Médiane

La position médiane $m(u)$ d'un sommet $u \in L_k$ est fonction des sommets auxquels u est connecté sur les niveaux adjacents L_{k-1} et L_{k+1}. Soit $\{v_1, v_2, \ldots, v_p\} \in N(u)$ sur L_{k-1} et $\{w_1, w_2, \ldots, w_q\} \in N(u)$ sur L_{k+1}. L'ensemble des positions normalisées de ces voisins est défini par :

$$N(u) = \left(\frac{\pi_{k-1}(v_1)}{n_{k-1}}, \ldots, \frac{\pi_{k-1}(v_p)}{n_{k-1}}, \frac{\pi_{k+1}(w_1)}{n_{k+1}}, \ldots, \frac{\pi_{k+1}(w_q)}{n_{k+1}} \right) \tag{3.14}$$

L'ensemble $N(u)$ est ensuite trié par ordre croissant et la position médiane est le nombre $m(u) \in N(u)$ qui partage $N(u)$ en deux sous-groupes de même effectif si $N(u)$ possède un nombre d'éléments impair ou le premier nombre du deuxième sous-groupe de $N(u)$ si le nombre d'éléments est pair. Le nouvel ordre des sommets π_k^m sur L_k est donné en triant l'ensemble des valeurs des positions médianes : pour chaque sommet $u, v \in L_k, \pi_k^m(u) \geq \pi_k^m(v)$ si et seulement si $m(u) \geq m(v)$.

La figure 3.18 illustre un exemple d'application de cet opérateur. Le sommet u_1 a 5 voisins sur L_{k-1} qui possède 8 sommets et 3 voisins sur L_{k+1} qui possède 6 sommets. L'ensemble trié $N(u_1)$ des différentes positions est donc (partie gauche de la figure) :

$$N(u_1) = \left(\frac{1}{8}, \frac{3}{8}, \frac{3}{6}, \frac{5}{8}, \frac{6}{8}, \frac{5}{6}, \frac{6}{6}, \frac{8}{8} \right) \Rightarrow m(u_1) = \frac{6}{8}$$

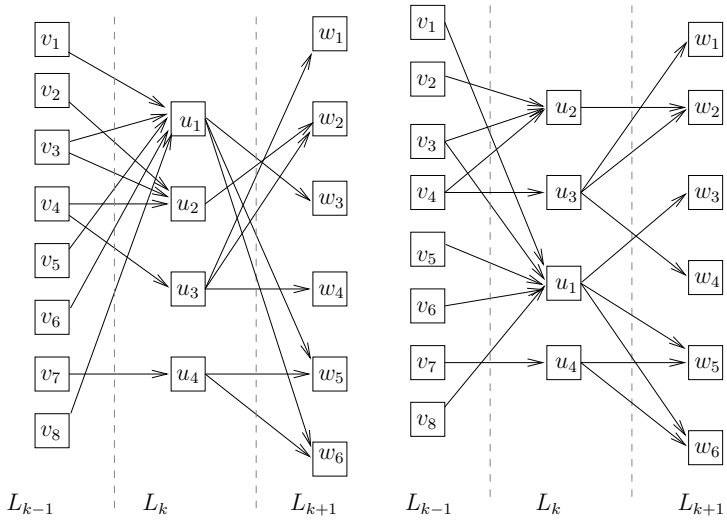
Pour les autres sommets on obtient :

$$N(u_2) = \left(\frac{2}{8}, \frac{2}{6}, \frac{3}{8}, \frac{4}{8}, \frac{2}{6} \right) \Rightarrow m(u_2) = \frac{3}{8}$$

$$N(u_3) = \left(\frac{1}{6}, \frac{2}{6}, \frac{3}{6}, \frac{4}{8} \right) \Rightarrow m(u_3) = \frac{3}{6}$$

$$N(u_4) = \left(\frac{5}{6}, \frac{7}{8}, \frac{6}{6} \right) \Rightarrow m(u_4) = \frac{7}{8}$$

Le nouvel ordre des sommets sur L_k est donc : $\pi_k = \{u_2, u_3, u_1, u_4\}$.



(a) Tracé original avec 27 croisements.

(b) Tracé avec permutation des sommets du niveau L_k selon leur position médiane. Il reste 9 croisements.

FIG. 3.18 – Exemple d'application de l'opérateur de la médiane.

O_3 : Barycentre

Le barycentre $b(u)$ du sommet $u \in L_k$ est défini par la moyenne arithmétique des positions normalisées des voisins de u sur L_{k-1} et L_{k+1} :

$$b(u) = \frac{\sum_{v_i} \frac{\pi_{k-1}(v_i)}{n_{k-1}} + \sum_{w_i} \frac{\pi_{k+1}(w_i)}{n_{k+1}}}{n_{k-1} + n_{k+1}} \tag{3.15}$$

Comme pour la médiane, le nouvel ordre des sommets π_k^b est déduit de l'ensemble des barycentres : $\forall u, v \in L_k, \pi_k^b(u) \geq \pi_k^b(v)$ si et seulement si $b(u) \geq b(v)$.

3.6.2 Résultats

Pour chaque graphe G de Δ', la solution dont le nombre de croisements d'arcs est minimal parmi les 5000 descentes est noté $\widehat{D_G}$. Du fait de l'ordre important des GP, nous ne pouvons bien sûr pas affirmer que cette valeur est l'optimum global. La distribution du nombre de descentes convergeant sur une solution $\widehat{D_G}$ est donnée par la figure 3.19. Le taux moyen de descentes convergeant sur une telle solution est relativement faible (28.6% avec un

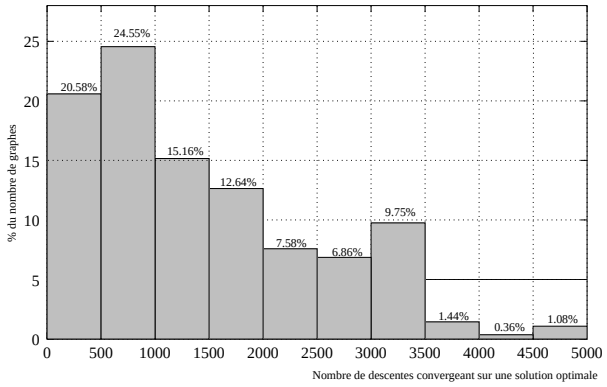


FIG. 3.19 – Distribution du nombre de descentes convergeant sur une solution $\widehat{D_G}$.

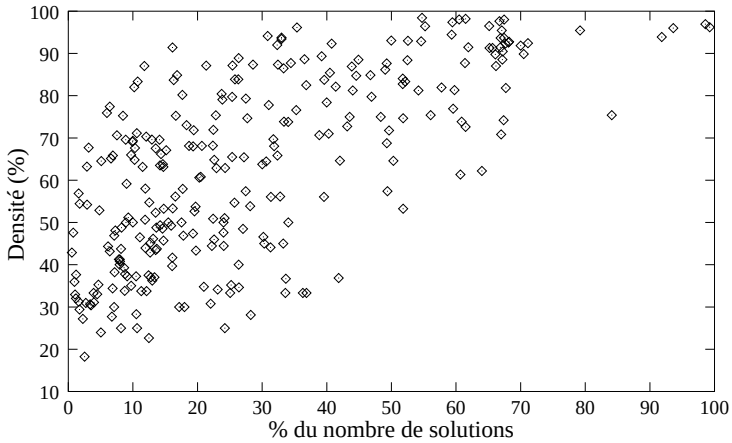
écart-type de 0.21) mais des différences importantes existent : pour 45% des graphes, plus de 80% des descentes conduisent à une solution non-optimale alors que pour 19% des graphes plus de la moitié des descentes convergent sur une solution $\widehat{D_G}$.

Pour le tracé des graphes, un coefficient de densité, fonction du nombre d'arcs du graphe, est souvent utilisé pour discriminer des classes. La densité d'un graphe est le ratio entre son nombre d'arcs m et le nombre total d'arcs $m_{\max}$ du graphe complet du même ordre. Le nombre maximal d'arcs d'un graphe en niveaux est atteint si un sommet est connecté à l'ensemble de ses voisins sur le niveau suivant :

$$m_{\max} = \sum_{k=2}^{h} n_{k-1} \times n_k \tag{3.16}$$

avec les niveaux numérotés de 1 à h. La densité d'un graphe en niveaux est donc : $d(G) = m/m_{\max}$.

Pour des graphes dont les densités sont proches des extrêmes (qui correspondent respectivement à des arbres et au graphe entièrement connecté), le problème de tracé est simple puisque peu de permutations sont possibles. Mais la situation est plus compliquée pour des valeurs intermédiaires. La relation entre le pourcentage de descentes convergeant sur une solution $\widehat{D_G}$ (abscisses) et la densité des graphes (ordonnées) est décrite par la figure 3.20. La corrélation linéaire entre ces deux variables est non nulle ($\rho = 0.35$). Ce résultat confirme que la densité est un facteur important pour la discrimination. Cependant, comme ici la dispersion est grande, il faudrait certainement compléter l'explication par d'autres facteurs.



Chaque point du graphique correspond à un graphe de Δ'.

FIG. 3.20 – Comparaison entre la densité des graphes et le pourcentage de solutions $\widehat{D_G}$ pour les descentes multiples.

3.7 Conclusion

Nous avons présenté dans ce chapitre une étude des paysages de recherche associés au tracé de graphes en niveaux. Pour faciliter cette étude, nous avons modélisé ces paysages par un graphe en niveaux appelé graphe-paysage. Cette modélisation implique que, contrairement à un problème continu, les intersections des bassins d'attraction des optima locaux et globaux peuvent être non vides. Cette caractéristique particulière entraîne une certaine complexité à trouver les optima globaux dans les paysages de recherche que nous avons confirmé et illustré par une étude statistique sur un ensemble de graphes de petites tailles. En complément avec une étude des paysages pour des graphes de grandes tailles nous avons montré que :

- les paysages sont fortement multimodaux,
- il existe des différences, parfois importantes, entre la qualité des optima globaux et locaux,
- certains optima locaux peuvent être de mauvaise qualité.

Nous avons aussi montré que la complexité du problème varie beaucoup en fonction de la densité du graphe mais utilisée seule, cette grandeur n'est pas suffisante pour constituer des classes de graphes ayant des propriétés communes.

Chapitre 4

Un algorithme génétique hybridé pour le tracé de graphes en niveaux

Sommaire

La complexité des paysages de recherche mise en évidence dans le chapitre précédent, ainsi que les comparaisons expérimentales prometteuses évoquées dans l'état de l'art nous ont incité à privilégier comme cadre de développe-

ment d'une méthode de résolution celui des métaheuristiques et plus précisément celui des algorithmes génétiques [11].

Lorsque le paysage de recherche ne présente pas de propriétés prototypiques telles que celles illustrées au paragraphe 3.3.2 page 49, on sait que la difficulté du problème à résoudre est difficile à estimer ; on peut se référer par exemple aux travaux de Mitchell *et al.* [98] qui montrent qu'un AG peut être efficace sur un paysage de recherche jugé *a priori* difficile ou au contraire qu'il peut avoir des performances médiocres sur un autre paysage jugé *a priori* facile. Le choix d'une métaheuristique s'avère donc souvent bien délicat [67]. Cependant, pour notre problématique, les algorithmes génétiques possèdent quelques caractéristiques qui les rendent *a priori* intéressants notamment le fait qu'ils permettent de réaliser un bon compromis entre l'exploration globale du paysage de recherche et l'exploitation du voisinage d'une solution particulière et qu'ils génèrent généralement des solutions valides pour le problème à traiter à chaque étape. Le processus de recherche peut donc être arrêté à tout moment.

Différents travaux ont tenté de classifier les problèmes selon la méthode de résolution la plus efficace [65, 99, 121, 120]. Il en ressort notamment que les algorithmes génétiques donnent de bons résultats quand les grimpeurs stochastiques échouent en raison de la présence de nombreux optima locaux [113, 120]. De plus les AG sont plus efficaces sur un paysage de recherche de grande taille grâce à leur capacité à sélectionner les parties du paysage qui contiennent des solutions de bonne qualité [51, 119].

De plus, les résultats obtenus préalablement par une approche évolutionnaire dans un autre contexte applicatif, celui de la fouille de règles d'association [86], nous ont conduit à approfondir cette voie. L'apport spécifique de notre travail est double. D'une part, nous montrons comment, associée à des opérateurs génétiques adaptées au problème, une hybridation avec une descente locale permet d'améliorer significativement les résultats. Une comparaison avec des descentes multiples et la méthode basée sur une recherche Tabou –dont la supériorité sur les méthodes déterministes a déjà été publiée par Laguna *et al.* en 1997 [83]– présentée dans le chapitre précédent, confirme l'intérêt de notre approche.

Ce chapitre est organisé comme suit. Après un bref rappel des principes de bases des AG et du vocabulaire associé, nous décrivons en 4.2 les différents opérateurs développés pour l'Algorithme Génétique Hybridé (AGH). Le paragraphe 4.3 présente une validation expérimentale sur 180 graphes en niveaux de taille standard engendrés de manière aléatoire selon deux critères (nombre de niveaux et densité des arcs). La validation de AGH porte sur 3 points principaux :

– la comparaison des différentes stratégies de recherche locale (para-

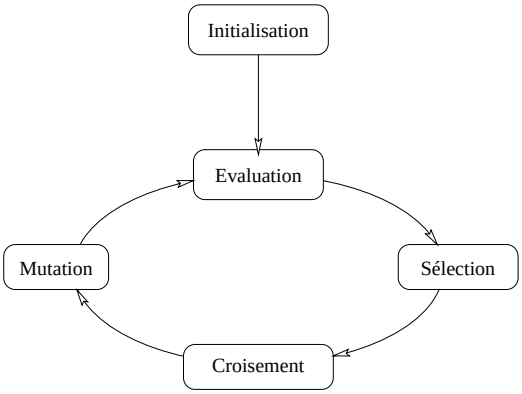


FIG. 4.1 – Principe général d'un algorithme génétique.

graphe 4.4),
- l'influence de chacun des opérateurs (paragraphe 4.5)
- et la comparaison avec les descentes en parallèle et la recherche Tabou (paragraphe 4.6).

4.1 Principe de base des AG

Introduits par J. Holland dans les années 70 [62] et rendus populaires en optimisation, par D. Goldberg [51], les algorithmes génétiques sont basés sur une métaphore des mécanismes simplifiés de la sélection naturelle et de la génétique de l'évolution néo-darwinienne.

Brièvement, le schéma général d'un AG (figure 4.1) consiste à faire évoluer un ensemble de solutions potentielles au problème, appelé *population*. Chaque solution –ou *individu*– est composée de différent *gènes*. Les individus sont *évalués* par une *fonction d'adaptation* qui mesure leur adéquation avec la solution idéale du problème traité. Les individus pour lesquels la fonction d'adaptation donne les meilleurs résultats sont *sélectionnés* pour la phase de *reproduction*. La reproduction permet de combiner plusieurs individus par *croisements* en vue de l'obtention espérée d'un meilleur individu. La phase de *mutation* permet de modifier localement un individu en changeant la valeur d'un gène. L'ensemble de ces étapes, appelé *génération,* est répété tant que la population ne satisfait pas certains critères de convergence (algorithme 4.1).

De nombreuses modifications de ce schéma de base ont été proposées et les AG constituent maintenant une classe de métaheuristiques dont l'intérêt a été expérimentalement montré pour de nombreux problèmes d'optimisation combinatoire (voir par exemple la série d'ouvrages *Foundations of Genetic*

Algorithme 4.1 Schéma de base d'un algorithme génétique

Entrée : Une fonction f à optimiser
Sortie : Un optimum de f

Début
Initialiser I individus de la population de départ P avec des solutions de f
Tant que critère de fin non atteint **faire**
  Évaluer chaque individu de P selon f
  Pour $j = 0, \ldots, (I/2 - 1)$ **faire**
    Sélectionner deux individus dans P
    Croiser ces deux individus pour en obtenir deux nouveaux
    Muter les deux nouveaux individus
    Conserver ces individus dans *population_ courante*
    $j \leftarrow j + 1$
  Fin Pour
  Remplacer les individus de P par ceux de *population_ courante*
Fin Tant que
Sélectionner l'individu de P qui optimise le mieux f
Fin

Algorithms publiée par Elsevier qui regroupe les meilleurs articles du colloque du même nom qui a lieu tous les deux ans [68]).

4.2 L'algorithme AGH pour le tracé de graphes en niveaux

On considère dans la suite un graphe en niveaux G. On rappelle que notre problème de tracé consiste à trouver, parmi l'ensemble des ordonnancements Π, celui qui minimise le nombre de croisements d'arcs $c(\Pi)$. Suivant le schéma de Sugiyama présenté au paragraphe 3.2 page 40, la suppression des cycles et le placement des sommets dans les niveaux sont réalisés au préalable avec des algorithmes présentés par Minoux et Bartnick [97] et déjà utilisés dans la thèse de R Lehn [86]. L'algorithme génétique a ici pour objectif la réduction du nombre de croisements d'arcs. Il suit le principe général des AG avec deux changements importants (algorithme 4.2) :

1. le croisement est composé de deux opérateurs adaptés au tracé de graphes en niveaux : un croisement inter-niveaux qui s'applique sur des niveaux entiers et un croisement intra-niveaux qui combine des

Algorithme 4.2 Algorithme génétique hybridé

Entrée : un graphe en niveaux G
Sortie : le tracé $\widehat{\Pi}$ de G qui minimise le nombre de croisements d'arcs $c(\Pi)$

Début
Initialiser avec des tracés aléatoires les I individus de la population de départ P
Tant que *critère_de_fin* non atteint **faire**
  Évaluer chaque individu de P
  Pour i=0,...,$(I/2-1)$ **faire**
    Sélectionner deux individus $g(\Pi)$ et $g(\Pi')$ dans P
    Appliquer le croisement intra-niveaux sur $g(\Pi)$ et $g(\Pi')$
      pour créer $g(\Pi_i)$ et $g(\Pi'_i)$
    Appliquer le croisement inter-niveaux sur $g(\Pi_i)$ et $g(\Pi'_i)$
      pour créer $g(\Pi_c)$ et $g(\Pi'_c)$
    Appliquer la mutation sur $g(\Pi_c)$ puis sur $g(\Pi'_c)$
    Appliquer la recherche locale sur $g(\Pi_c)$ puis sur $g(\Pi'_c)$
    Ajouter $g(\Pi_c)$ et $g(\Pi'_c)$ à *population_courante*
    $i \leftarrow i+1$
  Fin pour
  Remplacer les individus de P par ceux de *population_courante*
Fin Tant que
Choisir le tracé $\widehat{\Pi}$ qui minimise $c(\Pi)$ dans P
Fin

niveaux entre eux,

2. la phase de mutation est suivie d'une phase d'hybridation réalisée par une recherche locale selon un principe de descente.

4.2.1 Codage des solutions et fonction d'adaptation

Le problème pouvant se ramener à un problème de permutation, il était assez naturel de baser le codage des individus sur une représentation ordinale. Le codage des individus (ici les tracés Π de G) décrit la position des sommets sur chaque niveau les uns par rapport aux autres de L_1 à L_h ; pour chaque niveau L_k, on code le sommet présent à chaque position de 1 à n_k. Le génotype g associé à Π est défini par :

$$g(\Pi) = (\sigma_1(1), \ldots, \sigma_1(n_1), \sigma_2(1), \ldots, \sigma_2(n_2), \ldots, \sigma_h(1), \ldots, \sigma_h(n_h))$$

où $\sigma_k(i)$ indique le sommet qui se trouve à la position i sur le niveau k.

La fonction d'adaptation utilise un principe similaire à celui de la *normalisation linéaire*[1] introduit par Davis [26] et déjà utilisée pour du tracé de graphes non orientés [38]. Le principe est que le meilleur individu reçoit la valeur maximale d'adaptation (par ex. 100) et les autres individus reçoivent des valeurs inférieures proportionnelles (par ex. 98, 94, ...). L'écart entre les individus est déterminé par la fonction d'adaptation. Nous n'utilisons pas une fonction linéaire pour déterminer cet écart mais une adaptation de la fonction $f\left(g\left(\Pi\right)\right) = 2^{-c(\Pi)}$ afin qu'elle soit facilement calculable.

4.2.2 Opérateurs de croisements

Les opérateurs de croisements reprennent d'un point de vue opérationnel le "principe des briques de base[2]" : la combinaison de deux parties du tracé bien adaptées peut produire un nouveau tracé encore mieux adapté [52]. Cette combinaison illustrée sur la figure 4.2 est obtenue à partir d'un croisement entre les niveaux d'un graphe (croisement inter-niveaux) ou directement dans les niveaux (croisement intra-niveaux). Le croisement intra-niveaux est d'abord appliqué avec une probabilité p_{intra}, puis le croisement inter-niveaux est appliqué avec une probabilité p_{inter} (algorithme 4.3).

Croisement inter-niveaux

Cet opérateur est une adaptation du croisement à point-unique entre les niveaux. Il agit comme une coupe verticale dans le tracé. Soit i un nombre aléatoire compris dans l'intervalle $\{1, 2, \ldots, h\}$. Le résultat du croisement inter-niveaux entre deux parents $g\left(\Pi\right) = \left(\sigma_1(1), \ldots, \sigma_h\left(n_h\right)\right)$ et $g\left(\Pi'\right) = \left(\sigma'_1(1), \ldots, \sigma'_h\left(n_h\right)\right)$, donne deux nouveaux enfants :

$$\left(\sigma_1(1), \ldots, \sigma_{i-1}\left(n_{i-1}\right), \sigma'_i(1), \ldots, \sigma'_h\left(n_h\right)\right)$$

et

$$\left(\sigma'_1(1), \ldots, \sigma'_{i-1}\left(n_{i-1}\right), \sigma_i(1), \ldots, \sigma_h\left(n_h\right)\right)$$

Croisement intra-niveaux

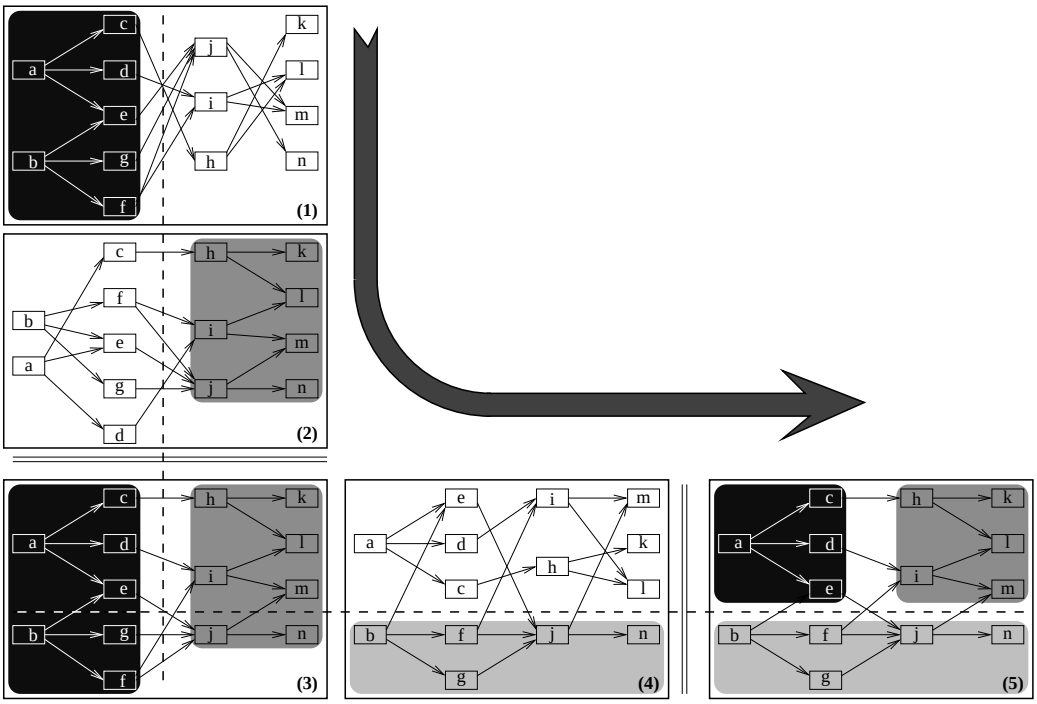
Combiner des briques de base au sein des niveaux ajoute une difficulté bien connue pour des codages ordinaux qui est de définir un opérateur de croisement qui garantisse que le résultat soit toujours une solution correcte [70, 139]. Le croisement intra-niveaux est une généralisation de *l'order crossover 1* [139] définit pour des permutations multiples.

[1] *linear normalization*
[2] *Building blocks*

Algorithme 4.3 Application des opérateurs de croisements.

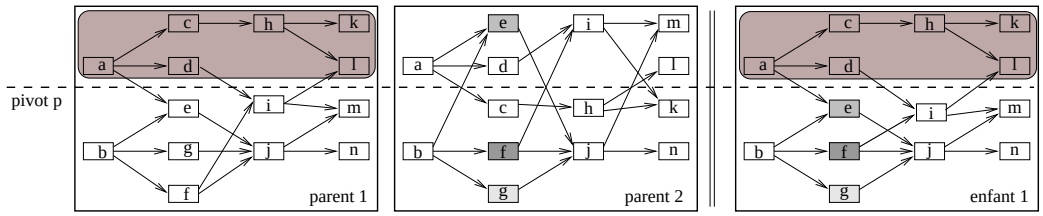
Entrée : deux individus $g(\Pi)$ et $g(\Pi')$
Sortie : les individus $g(\Pi_c)$ et $g(\Pi'_c)$ obtenus après applications éventuelles des opérateurs

Début
Si le test sur p_{intra} est validé **alors**
Appliquer le croisement intra-niveaux sur $g(\Pi)$ et $g(\Pi')$ pour créer $g(\Pi_i)$ et $g(\Pi'_i)$
Sinon
$g(\Pi_i) \leftarrow g(\Pi)$
$g(\Pi'_i) \leftarrow g(\Pi')$
Fin si
Si le test sur p_{inter} est validé **alors**
Appliquer le croisement inter-niveaux sur $g(\Pi_i)$ et $g(\Pi'_i)$ pour créer $g(\Pi_c)$ et $g(\Pi'_c)$
Sinon
$g(\Pi_c) \leftarrow g(\Pi_i)$
$g(\Pi'_c) \leftarrow g(\Pi'_i)$
Fin si
Retourner $g(\Pi_c)$ et $g(\Pi'_c)$
Fin



– *Combinaison de la partie gauche du tracé (1) avec la partie droite du tracé (2) pour créer (3) : croisement inter-niveaux.*
– *Combinaison de (3) et (4) pour créer (5) qui est un tracé idéal : croisement intra-niveaux.*
– *Ces différentes combinaisons permettent l'émergence de briques de bases qui permettront de créer des tracés bien adaptés.*

FIG. 4.2 – Application des opérateurs de croisements sur différents sous-graphes.



1. *Définition d'un point de coupure p au sein de chaque niveau (pivot) dont la position est normalisée en fonction du nombre de sommets par niveau.*
2. *La partie de "parent 1" au dessus du pivot est recopiée directement dans le premier enfant ("enfant 1").*
3. *Les sommets manquants pour "enfant 1" (par exemple* e, f *et* g *pour le niveau 2) sont ajoutés en conservant l'ordre qu'ils avaient dans "parent 2".*
4. *Ce processus est inversé pour créer le deuxième enfant (non représenté ici).*

FIG. 4.3 – Exemple d'application du croisement intra-niveaux pour la génération du premier enfant.

Un pivot p est aléatoirement choisi et sa position est normalisée en fonction de la cardinalité de chacun des niveaux. Au dessus de ce pivot, la position des sommets du premier parent est conservée. En dessous, les sommets manquants sont ajoutés en conservant leurs ordres dans le deuxième parent. Ce processus est inversé pour créer le deuxième enfant (voir la figure 4.3).

La position du pivot p pour un niveau L_k est choisie de façon uniforme dans $\{1, 2, \ldots, n_k\}$. Le résultat pour chacun des enfants pour un niveau L_k entre $g(\Pi)$ et $g(\Pi')$ est :

$$(\sigma_k(1), \ldots, \sigma_k(p), \rho'_k(1), \ldots, \rho'_k(n_k - p))$$

et

$$(\sigma'_k(1), \ldots, \sigma'_k(p), \rho_k(1), \ldots, \rho_k(n_k - p))$$

avec $\rho'_k(i)$ (resp. $\rho_k(i)$) le $i^{\text{ème}}$ sommet de $(\sigma'_k(1), \sigma'_k(2), \ldots, \sigma'_k(n_k))$ (resp. $(\sigma_k(1), \sigma_k(2), \ldots, \sigma_k(n_k))$) qui n'est pas présent dans $(\sigma_k(1), \sigma_k(2), \ldots, \sigma_k(n_k))$ (resp. $(\sigma'_k(1), \sigma'_k(2), \ldots, \sigma'_k(n_k))$).

4.2.3 Hybridation par une recherche locale

Une stratégie de recherche locale qui globalement permet d'améliorer les résultats, sans trop détériorer le temps de calcul, nécessite un grand nombre d'essais et d'erreurs sur une base de tests importante. Dans le cas d'une optimisation locale, une approche, souvent privilégiée, est la descente en profondeur quand le voisinage d'une solution peut être facilement défini.

Algorithme 4.4 Principe de la stratégie de recherche locale

Entrée : un individu $g(\Pi)$
Sortie : l'individu $g(\Pi)$ obtenus après les applications éventuelles des opérateurs

Début
Pour chaque niveau L_k de $g(\Pi)$ **faire**
  Si le test sur p_{inv} est validé **alors**
    Appliquer O_1 sur L_k pour créer L_{k_i}
    Remplacer L_k par L_{k_i} dans $g(\Pi)$
  Fin si
Fin Pour
Pour chaque niveau L_k de $g(\Pi)$ **faire**
  Si le test sur p_{bar} est validé **alors**
    Appliquer O_3 sur L_k pour créer L_{k_b}
    Remplacer L_k par L_{k_b} dans $g(\Pi)$
  Fin si
Fin Pour
Pour chaque niveau L_k de $g(\Pi)$ **faire**
  Si le test sur p_{med} est validé **alors**
    Appliquer O_2 sur L_k pour créer L_{k_m}
    Remplacer L_k par L_{k_m} dans $g(\Pi)$
  Fin si
Fin Pour
Retourner $g(\Pi)$
Fin

Pour des tracés de graphes, différentes métriques peuvent être envisagées pour définir une topologie [14]. Mais, bien souvent le calcul de ces métriques est coûteux en temps de calculs ; le choix de la plus adaptée est encore un problème ouvert. Une autre solution, comme pour l'analyse des graphes-paysages, est de recourir à des heuristiques conçues spécifiquement pour le problème à traiter. C'est le choix que nous avons fait ici.

La stratégie de recherche locale utilisée combine une application sur chaque niveau du graphe des trois opérateurs introduit pour l'analyse des graphes-paysages (paragraphe 3.6.1 page 61). L'inversion de sommets adjacents est appliquée avec une probabilité p_{inv}, l'opérateur de médiane avec une probabilité p_{med} et le barycentre avec une probabilité p_{bar} (algorithme 4.4).

Algorithme 4.5 Opérateur de mutation

Entrée : un individu $g(\Pi)$
Sortie : l'individu $g(\Pi)$ obtenu après une application éventuelle de l'opérateur

Début
Pour chaque niveau L_k de $g(\Pi)$ **faire**
  Si le test sur p_{mut} est validé **alors**
    Inverser deux sommets choisis aléatoirement dans L_k pour créer $L_{k_{mut}}$
    Remplacer L_k par $L_{k_{mut}}$ dans $g(\Pi)$
  Fin si
Retourner $g(\Pi)$
Fin

4.2.4 Mutation et critère de fin

L'opérateur de mutation consiste simplement à inverser deux sommets d'un même niveau choisis aléatoirement. Il est appliqué successivement sur chaque niveau avec une probabilité p_{mut} (algorithme 4.5).

L'algorithme s'arrête après un certain nombre de générations sans amélioration du meilleur tracé trouvé. Ce nombre a été fixé expérimentalement (voir paragraphe suivant).

4.3 Validation expérimentale des paramètres et des opérateurs

Les expérimentations numériques ont été effectuées sur un ensemble de 180 graphes en niveaux connexes engendrés aléatoirement qui ne soient pas des arbres. Afin d'être compatible avec les expérimentations menées pour d'autres métaheuristiques, le générateur fonctionne comme celui précédemment présenté par Laguna *et al.* [83]. Trois paramètres peuvent être modifiés : le nombre de niveaux, le nombre de sommets par niveau et la densité du graphe. Le générateur a été utilisé pour créer 20 instances de chaque combinaison possible entre 4, 8 ou 12 niveaux avec une densité de 0.3, 0.5 ou 0.7. Le nombre de sommets par niveaux est aléatoirement choisi entre 5 et 15. Ces caractéristiques sont représentatives d'applications réelles dans lesquelles les sommets sont représentés par des boîtes et leur nombre peut rarement dépasser 70 ou 80 sur une feuille ou un écran de taille standard.

Pour comparer les différentes approches, nous calculons le "pourcentage

de meilleures solutions trouvées" : pour chaque exécution de k algorithmes $A_1, \ldots, A_k$ sur un même graphe, on compte le nombre d'exécutions pour lesquelles chaque A_i a été le seul à trouver la meilleure solution. Nous considérons aussi spécifiquement les cas d'égalités où plusieurs algorithmes trouvent la meilleure solution. Cette information souvent passée sous silence dans la littérature s'est ici révélée importante.

Pour mesurer la stabilité des résultats, AGH a été exécuté 100 fois pour chaque graphe (18000 exécutions au total). Les calculs ont été effectués sur un ordinateur PC de type AMD Athlon MP 2400+ sous Linux. AGH est codé en C.

4.3.1 Estimation des probabilités d'applications des opérateurs

Deux stratégies de recherche locale ont été envisagées pour AGH : RL1 qui suit une stratégie d'exploration locale du paysage de recherche et RL2 qui suit une stratégie d'optimisation locale. Pour RL1, il n'y a pas d'évaluation intermédiaire des solutions calculées par les opérateurs, elle est effectuée lors de la phase d'évaluation de AGH (algorithme 4.4). Pour RL2, le tracé produit après chaque application des opérateurs n'est conservé que si le nombre de croisements diminue (algorithme 4.6). Dans la suite, AGH (resp. AGH2) est l'AG hybridé associé à RL1 (resp. RL2).

Les résultats des calculs donnés dans les tableaux 4.1 et 4.2 montrent que la combinaison des meilleurs paramètres est proche pour AGH et AGH2. Des petites variations autour de ces valeurs ne modifient pas de façon significative la qualité des résultats ou le temps de calcul. Les paramètres retenus pour AGH (resp. AGH2) sont : $p_{mut} = 0.02$, $p_{intra} = p_{inter} = 0.2$, $p_{inv} = 0.05$ et $p_{bar} = p_{med} = 0.2$ (resp. $p_{mut} = 0.02$, $p_{intra} = p_{inter} = 0.25$, $p_{inv} = 0.1$ et $p_{bar} = p_{med} = 0.3$).

4.3.2 Taille de la population et critère d'arrêt

La taille de la population a été fixée à 100 individus. L'algorithme s'arrête quand aucune amélioration de la meilleure solution trouvée ne se produit pendant 100 générations consécutives. Le tableau 4.3 montre évidemment qu'une population plus grande et un plus grand nombre de générations avant arrêt donnent de meilleurs résultats. Mais le choix d'une taille de population est un compromis entre la qualité des solutions et le temps de calcul qui est un paramètre critique en particulier pour les extensions au tracé dynamique.

Algorithme 4.6 Stratégie d'optimisation locale RL2

Entrée : un individu $g(\Pi)$
Sortie : l'individu $g(\Pi)$ après les applications éventuelle des opérateurs
Variable : un individu $g(\Pi_c)$

Début
$g(\Pi_c) \leftarrow g(\Pi)$
Pour chaque niveau L_k de $g(\Pi_c)$ **faire**
  Si le test sur p_{inv} est validé **alors**
    Appliquer O_1 sur L_k pour créer L'_k
    Remplacer L_k par L'_k dans $g(\Pi_c)$
  Fin si
Fin Pour
Si $c(\Pi_c) > c(\Pi)$ **alors**
  $g(\Pi_c) \leftarrow g(\Pi)$
Sinon
  $g(\Pi) \leftarrow g(\Pi_c)$
Fin si
Pour chaque niveau L_k de $g(\Pi_c)$ **faire**
  Si le test sur p_{bar} est validé **alors**
    Appliquer O_3 sur L_k pour créer L'_k
    Remplacer L_k par L'_k dans $g(\Pi_c)$
  Fin si
Fin Pour
Si $c(\Pi_c) > c(\Pi)$ **alors**
  $g(\Pi_c) \leftarrow g(\Pi)$
Sinon
  $g(\Pi) \leftarrow g(\Pi_c)$
Fin si
Pour chaque niveau L_k de $g(\Pi_c)$ **faire**
  Si le test sur p_{med} est validé **alors**
    Appliquer O_2 sur L_k pour créer L'_k
    Remplacer L_k par L'_k dans $g(\Pi_c)$
  Fin si
Fin Pour
Si $c(\Pi_c) < c(\Pi)$ **alors**
  $g(\Pi) \leftarrow g(\Pi_c)$
Fin si
Retourner $g(\Pi)$
Fin

p_{inv}	0.05	0.05	0.1	0.1	0.1
p_{bar}	0.1	0.2	0.1	0.2	0.3
p_{med}	0.1	0.2	0.1	0.2	0.3
% meilleures solutions	33.5	41.17	28.4	35.7	40.42
Temps de calcul moyen (en sec.) (écart-type)	2.46 (2.56)	2.58 (2.62)	2.54 (2.7)	2.63 (2.7)	2.78 (2.8)

a. Résultats avec différentes probabilités des opérateurs de recherche locale.

p_{intra}	0.2	0.25	0.3
p_{inter}	0.2	0.25	0.3
% meilleures solutions	48.15	48.4	47.9
Temps de calcul moyen (en sec.) (écart-type)	2.58 (1.78)	2.57 (1.76)	2.58 (1.73)

b. Résultats avec différentes probabilités des opérateurs de croisements.

Les résultats incluent les cas d'égalité. Ceci explique que la somme des lignes soit supérieure à 100%.

TAB. 4.1 – Distribution du % de meilleures solutions trouvées et temps moyen de calcul pour différentes distributions des paramètres pour AGH.

p_{inv}	0.05	0.05	0.1	0.1	0.1
p_{bar}	0.1	0.2	0.1	0.2	0.3
p_{med}	0.1	0.2	0.1	0.2	0.3
% meilleures solutions	23.05	33.66	21.83	30.81	38.71
Temps de calcul moyen (en sec.) (écart-type)	3.66 (3.17)	3.70 (3.13)	3.64 (3.14)	3.70 (3.12)	3.80 (3.16)

a. Résultats avec différentes probabilités des opérateurs de recherche locale.

p_{intra}	0.2	0.25	0.3
p_{inter}	0.2	0.25	0.3
% meilleures solutions	41.76	72.87	41.47
Temps de calcul moyen (en sec.) (écart-type)	3.70 (3.13)	3.69 (3.12)	3.69 (3.12)

b. Résultats avec différentes probabilités des opérateurs de croisements.

Les résultats incluent les cas d'égalité. Ceci explique que la somme des lignes soit supérieure à 100%.

TAB. 4.2 – Distribution du % de meilleures solutions trouvées et temps moyen de calcul pour différentes distributions des paramètres pour AGH2.

Nombre de générations	50	100	150	200	300
% meilleures solutions	28.61	35.1	39.11	42.18	47.16
Temps de calcul moyen (en sec.)	1.45	2.57	3.64	4.66	6.72

a. Nombre de générations consécutives sans amélioration avant arrêt

Nombre d'individus	50	100	150	200	250	300
% meilleures solutions	24.77	31.06	35.51	39.15	41.3	43.97
Temps de calcul moyen (en sec.)	1.33	2.57	3.79	5.03	6.19	7.29

b. Taille de la population

Les résultats incluent les cas d'égalité. Ceci explique que la somme des lignes soit supérieure à 100%.

TAB. 4.3 – Distribution du % de meilleures solutions trouvées pour (a) différents nombres de générations et (b) différentes tailles de population.

	AGH	AGH2	AGH=AGH2
$d(G) = 0.3$	64.02	23.55	12.43
$d(G) = 0.5$	60.43	25.75	13.82
$d(G) = 0.7$	54.8	26.4	18.8

AGH=AGH2 signifie que les deux méthodes ont trouvé une solution avec le même nombre de croisements.

TAB. 4.4 – Comparaison du % de meilleures solutions trouvées pour AGH et AGH2.

4.4 Comparaison des stratégies de recherche locale

Le tableau 4.4 montre que AGH est nettement meilleur que AGH2 quelle que soit la densité des graphes. Pour comparer la qualité des solutions obtenues par les métaheuristiques, nous utilisons un test statistique [129]. Le test de Mann-Whitney ou test de Wilcoxon pour des échantillons appariés est ici bien approprié : il s'agit d'un test non paramétrique qui permet de comparer deux ensembles triés de valeurs en comparant la somme des rangs de ces valeurs. Si la valeur $p-value$ donnée par le résultat du test est petite alors l'hypothèse H_0 de différence significative des échantillons est vérifiée. La $p-value$ donnée par le test sur le nombre de croisements est 7.885×10^{-10} pour un risque standard $\alpha = 0.05$. Donc le test confirme que AGH est meilleur que AGH2.

Le temps nécessaire pour effectuer le test d'arrêt est plus grand pour AGH2 (3.8 secondes) que pour AGH (2.6 secondes). Cette différence est due au coût engendré par l'évaluation des différentes solutions calculées par chacun des trois opérateurs de RL2. Quand on se limite à la première génération qui trouve la meilleure solution (sans les 100 générations supplémentaires de la condition d'arrêt), AGH effectue en moyenne 90.86 générations (écart-type de 58.63) en 2.36s et AGH2 effectue en moyenne 16.54 générations (écart-type de 8.74) en 0.59s. Ce dernier résultat s'explique par l'application de l'optimisation locale dans AGH2 qui réduit l'effet positif de la phase d'exploration de l'AG ; il converge sur une solution qui peut être de mauvaise qualité trop rapidement. En conséquence pour la suite, seulement AGH est utilisé.

4.5 Influence des différents opérateurs

Afin de mieux comprendre le fonctionnement de AGH, nous étudions l'influence de ses différents opérateurs sur la qualité des résultats. Tout d'abord nous vérifions l'influence de la recherche locale en comparant AGH avec une version sans la recherche locale. En effet, l'ajout d'une phase de recherche locale à un algorithme génétique classique a une influence sur trois caractéristiques : amélioration de la convergence pour trouver la meilleure solution, réduction du temps de calcul et de la variabilité des résultats inhérente à l'approche stochastique.

Puis, pour évaluer l'importance de la part génétique, nous comparons AGH avec une méthode basée sur des descentes stochastiques à partir d'un tracé initial aléatoire. Ces descentes sont basées sur la partie de recherche locale de l'algorithme. Nous étudions aussi le nombre de points du paysage de recherche évalués par chacune des méthodes et la qualité des optima obtenus par une des méthodes quand l'autre trouve une meilleure solution.

4.5.1 Opérateurs de recherche locale

Pour la comparaison entre AGH et AG (AGH sans la partie de recherche locale) la meilleure solution est obtenue par AGH dans environ 88.6% des cas (voir tableau 4.5 pour les détails). Cette différence de performance est confirmée par le test de Wilcoxon ($p - value < 2.2 \times 10^{-16}$ pour $\alpha = 0.05$). L'amélioration globale du temps de calcul donnée par $(t_{AG} - t_{AGH}) / t_{AG}$ est égale à 28.5% et augmente avec la densité : 21.1% pour $d(G) = 0.3$, 29.4% pour $d(G) = 0.5$ et 35.2% pour $d(G) = 0.7$. L'amélioration de la stabilité des solutions est donnée par un rapport entre les moyennes des écart-types du nombre de croisements obtenus pour les 100 exécutions de AGH pour chaque

	AGH	AG	AGH=AG
$d(G) = 0.3$	88.58	8.35	3.13
$d(G) = 0.5$	81.9	14.22	3.88
$d(G) = 0.7$	72.28	18.02	9.7

AGH=AG signifie que les deux méthodes ont trouvé une solution avec le même nombre de croisements.

TAB. 4.5 – % moyen de meilleures solutions trouvées entre AGH et AG.

graphe : $(\overline{\sigma}_{AG} - \overline{\sigma}_{AGH}) / \overline{\sigma}_{AG} = 53.2\%$.

4.5.2 Opérateurs génétiques

AGH a été comparé avec des descentes multiples (DM) dans laquelle la partie de recherche locale de AGH est exécutée seule plusieurs fois. Plus précisément à partir d'un arrangement initial aléatoire, RL1 est appliquée en suivant une stratégie de descente avec $p_{inv} = p_{med} = p_{bar} = 1$. La solution est conservée tant que le nombre de croisements diminue. La descente s'arrête quand aucune amélioration n'est plus possible.

Deux stratégies de descentes sont envisagées : –(DMa) une génération aléatoire de 100 descentes, valeur de référence souvent utilisée, et –(DMb) qui possède un critère d'arrêt égal au temps de convergence de AGH sur la meilleure solution.

Le tableau 4.6 montre que 100 descentes ne sont pas suffisantes pour obtenir de meilleurs résultats que AGH. Avec DMb, le nombre moyen d'arrangements de départ est d'environ 677.58. Pour cette dernière comparaison, la meilleure solution est plus souvent calculée par AGH quand $d(G) = 0.3$ et 0.5 pour des graphes comportant $h = 4$ ou $h = 8$ niveaux ou bien quand $d(G) = 0.7$ et $h = 4$. Mais pour les autres cas, quand les graphes sont les plus grands, la différence entre les méthodes n'est plus significative. Néanmoins, on peut quand même remarquer que pour $d(G) = 0.7$, 13.5% des meilleures solutions sont simultanément obtenues avec les deux approches.

4.5.3 Exploration du paysage de recherche

Une heuristique d'optimisation peut être considérée comme efficace si elle évalue le plus grand nombre de solutions intermédiaires possibles pendant un temps de calcul raisonnable et si la solution finale est de bonne qualité.

Sur l'ensemble de la base de 180 graphes, DMb effectue en moyenne 5.27 applications successives des trois opérateurs sur l'ensemble des niveaux avant

h	4	8	12	Moy.
AGH	52.9	48.4	44.5	48.6
DMa	3.85	6.2	6.45	5.5
DMb	10.7	37.3	43.25	30.41
AGH=DMa	3.55	0.45	0.5	1.5
AGH=DMb	4.45	2.15	3.2	3.27
DMa=DMb	9.5	4.8	1.9	5.4
AGH=DMa=DMb	15.05	0.7	0.2	5.32

a. Résultats pour $d(G) = 0.3$

h	4	8	12	Moy.
AGH	56.55	51.8	35.05	47.8
DMa	3.35	5.25	7.7	5.43
DMb	9.3	33.1	51.4	31.27
AGH=DMa	1.2	0.35	0.4	0.65
AGH=DMb	1.7	2.2	2.5	2.13
DMa=DMb	9.2	7	2.8	6.33
AGH=DMa=DMb	18.7	0.3	0.15	6.39

b. Résultats pour $d(G) = 0.5$

h	4	8	12	Moy.
AGH	41.3	38.95	32.75	37.67
DMa	3.05	6.45	8	5.83
DMb	6.4	46.9	54.75	36.02
AGH=DMa	2.85	0.1	0.15	1.03
AGH=DMb	2.8	1.25	1.55	1.87
DMa=DMb	9.2	6.1	7.75	6.02
AGH=DMa=DMb	34.4	0.25	0.05	11.56

c. Résultats pour $d(G) = 0.7$

La colonne AGH=DMa signifie que les deux approches ont trouvé une solution avec le même nombre de croisements.

TAB. 4.6 – Comparaisons du % moyen de meilleures solutions trouvées entre AGH, et les descentes multiples (DMa et DMb).

d'arriver à une solution finale. Donc le nombre total moyen d'évaluations pour DMb est $5.27 \times 677.58 = 3570.85$. Pour AGH le nombre moyen d'évaluations est simplement la taille de la population multipliée par le nombre moyen de générations : $100 \times 90.86 = 9086$. Donc pendant le même temps de calcul, AGH explore deux fois plus de solutions que DMb.

Quand la découverte d'un optimum global n'est pas garantie, la qualité des optima locaux calculés est un facteur important. Nous utilisons ici une adaptation de la formule de la hauteur relative vue précédemment (paragraphe 3.5 page 56). Quand la meilleure solution Π_M est trouvée par une métaheuristique M différente de AGH, on compare la qualité de cette dernière avec la qualité de la solution trouvé par AGH. La hauteur relative $h_{DMb}(\Pi_{AGH})$ de la solution Π_{AGH} trouvée par AGH quand DMb trouve la meilleure solution est définie par :

$$h_{DMb}(\Pi_{AGH}) = 1 - \frac{c(\Pi_{AGH}) - c(\Pi_{DMb})}{c(\Pi_{DMb})} \tag{4.1}$$

Si $h_{DMb}(\Pi_{AGH1})$ est proche de 1, alors la qualité du tracé peut-être considérée comme correcte[3]. Ceci est vrai en particulier pour des graphes de densité importante quand l'ajout de quelques croisements n'est pas forcément visuellement perceptible sur le tracé. Pour des graphes de densité 0.7, la distribution de $h_{DMb}(\Pi_{AGH})$ est concentrée sur l'intervalle [0.96,1].

Ces résultats montrent que même pour des densités importantes, AGH reste efficace même s'il ne trouve pas les meilleurs résultats. En revanche, pour les graphes d'ordres et de densités importants, il semblerait qu'une recherche basée sur une méthode stochastique, rapide en temps de calcul, qui permet une exploration à grande échelle du paysage de recherche donne de meilleurs résultats que AGH. Néanmoins, le nombre d'instances pour lesquelles ces faits ont été constatés est relativement limité (environ 20) et avant de pouvoir faire une conclusion correcte, des expérimentations supplémentaires doivent être menées. Ces expérimentations font l'objet d'une partie du prochain chapitre (chapitre 5, paragraphe 5.2 page 96) qui est consacré à deux extensions de AGH.

4.6 Comparaisons avec la recherche Tabou

Nous avons comparé AGH avec la méthode basée sur une recherche Tabou (TS) présentée au paragraphe 3.2.2 page 42 qui est connue pour être la

[3]Cette indice plus simple que celui introduit précédemment (chapitre 3, paragraphe 3.5 page 56) n'est valable que pour cet ensemble de graphes car $c(\Pi_{DMb})$ n'est jamais nul.

h	4	8	12	Moy.
AGH	100	100	87.5	95.83
TS	0	0	8.33	2.78
AGH=TS	0	0	4.17	1.39

a. Résultats pour $d(G) = 0.3$

h	4	8	12	Moy.
AGH	91.3	88.89	94.44	91.54
TS	4.35	11.11	5.56	7.01
AGH=TS	4.34	0	0	1.45

b. Résultats pour $d(G) = 0.5$

h	4	8	12	Moy.
AGH	79.17	77.78	83.33	80.09
TS	8.33	22.22	16.67	15.74
AGH=TS	12.5	0	0	4.17

c. Résultats pour $d(G) = 0.7$

AGH=TS signifie que les deux méthodes ont trouvé une solution avec le même nombre de croisements.

TAB. 4.7 – Comparaison entre AGH et TS. % moyen du nombre de meilleures solutions trouvées.

meilleure métaheuristique pour notre problème. Nous avons utilisé la version qui privilégie la qualité des solutions obtenues sans contrainte majeure de temps de calcul. Elle consiste à s'arrêter après 50 itérations sans amélioration du meilleur résultat (algorithme 4.7).

Le tableau 4.7 montre clairement que AGH est nettement plus efficace que TS et cette différence est confirmée par le test de Wilcoxon. Pour les rares cas où TS supplante AGH, le tableau 4.8 montre que les solutions trouvées par AGH sont très proches des meilleures solutions de TS. Au contraire, quand AGH trouve la meilleure solution, les solutions de TS peuvent être assez éloignées particulièrement pour des graphes de faibles densités.

4.7 Conclusion

Nous avons présenté dans ce chapitre un nouvel algorithme génétique hybridé pour le tracé de graphes en niveaux. Cet algorithme suit le principe général des AG avec deux spécificités :

Algorithme 4.7 Méthode de tracé avec une recherche Tabou [83].

Entrée : Un graphe G à tracer
Sortie : Un tracé de G contenu dans *meilleure_solution*

Début
Engendrer un tracé initial de G aléatoire dans *solution_courante*
meilleure_solution $\leftarrow$ *solution_courante*
$i \leftarrow 0$
Tant que $i < 50$ **faire**
  $m \leftarrow 0$
  Tant qu'il existe un niveau non tabou dans *solution_courante* **faire**
    Choisir un niveau L_k non tabou dans *solution_courante*
      en fonction de son importance à engendrer des croisements
    Trouver le meilleur optimum local o_l en permutant les sommets de L_k
    Si $c(o_l) < c(solution\_courante)$ **alors**
      solution_courante $\leftarrow o_l$
      meilleure_solution $\leftarrow$ *solution_courante*
      Marquer L_k tabou dans *solution_courante*
      Rendre L_{k-1} et L_{k+1} non tabou dans *solution_courante*
      $m \leftarrow 1$
    Sinon
      Marquer L_k tabou dans *solution_courante*
    Fin si
  Fin Tant que
  Si $m == 1$ **alors**
    $i \leftarrow 0$
  Sinon
    $i \leftarrow i + 1$
  Fin si
  Appliquer la phase de recherche locale autour de *solution_courante*
  Si une meilleure solution s_l est trouvée **alors**
    solution_courante $\leftarrow s_l$
    meilleure_solution $\leftarrow$ *solution_courante*
  Fin si
Fin Tant que
Afficher le tracé de G contenu dans *meilleure_solution*
Fin

$d(G)$	0.3	0.5	0.7
$h_{TS}(AGH)$	0.991	0.986	0.994
$h_{AGH}(TS)$	0.848	0.955	0.977

TAB. 4.8 – Hauteur relative moyenne $h_{TS}(AGH)$ des solutions trouvées par AGH quand TS trouve la meilleure solution, et hauteur relative moyenne $h_{AGH}(TS)$ des solutions trouvées par TS quand AGH trouve la meilleure solution.

1. utilisation de deux opérateurs de croisements adaptés spécifiquement pour le tracé de graphes,
2. hybridation par une phase de recherche locale pour réduire le temps de convergence et plus globalement le temps de calcul ainsi que la variabilité des solutions.

Nous avons montré que chaque opérateur de AGH est nécessaire afin d'obtenir des solutions de bonnes qualités. Les expérimentations ont montré que AGH surpasse les méthodes exactes classiques et donne des résultats de meilleurs qualités avec un temps de calcul plus faible que les autres métaheuristiques. Néanmoins, lorsque l'ordre et la densité des graphes deviennent importants, AGH semble donner des résultats comparables ou moins bons que la méthode de descentes. Pour essayer de mieux comprendre ce phénomène, ces deux méthodes sont comparées sur des graphes de grands tailles dans le chapitre suivant.

Chapitre 5

Extensions de l'algorithme génétique et de ses applications

Sommaire

Ce chapitre présente deux extensions de l'Algorithme Génétique Hybridé (AGH) développé au chapitre précédent. La première concerne le tracé dynamique où le graphe évolue sur une période de temps restreinte et la seconde porte sur la comparaison de AGH et des descentes multiples sur des graphes de grandes tailles.

Les expérimentations ont montré que AGH est un algorithme efficace pour réduire le nombre de croisements d'arcs pour des graphes dont l'ordre est compatible avec un affichage sur un support de taille standard. Dans le cadre de l'application en gestion des connaissances, et plus généralement pour tout processus de construction de graphes, il faut encore traiter l'aspect dynamique du processus de construction des modèles de connaissances : sur une période de temps restreinte (quelques secondes), entre deux étapes consécutives $t-1$ et t, l'utilisateur ajoute ou enlève des sommets et/ou des arcs au graphe. En plus de respecter les différentes contraintes de lisibilité, le tracé à l'instant t doit aussi conserver une certaine ressemblance avec le

tracé à $t-1$; ceci afin de minimiser l'effort cognitif nécessaire à l'utilisateur pour interpréter le nouveau tracé. Pour prendre en compte cette contrainte supplémentaire, les algorithmes évolutionnaires sont des méthodes bien adaptées pour résoudre des problèmes multiobjectifs [21]. En effet, il est souvent impossible de trouver simultanément les valeurs optimales des différents paramètres. Il faut donc se contenter d'une ou plusieurs solutions approchées qui réalisent un bon compromis entre les différentes valeurs des paramètres.

Essentiellement pour des raisons de complexité –et également de temps–, le problème multiobjectif associé n'a pas été abordé de front ici. La contrainte de proximité entre tracés consécutifs est initialement relaxée et est réintroduite sur la sous-population de AGH conduisant aux meilleurs résultats : au lieu de sélectionner le meilleur tracé calculé, on sélectionne parmi les meilleurs celui qui est le plus proche du tracé précédemment calculé. Les premières comparaisons qui sont présentées sur un simulateur de générations de graphes dynamiques montrent que cette voie est prometteuse eu égard à la qualité des solutions "optimales" obtenues par d'autres approches.

Les expérimentations menées sur AGH ont aussi montré que pour les graphes les plus grands, il semble que AGH perde de son efficacité au profit des descentes multiples. Pour mieux comprendre ce constat, nous avons effectué de nouvelles comparaisons sur un ensemble de graphes de grandes tailles (environ 550 sommets en moyenne avec un écart-type de 100 et environ 7500 arcs en moyenne avec un écart-type de 1700) entre ces deux méthodes. La représentation en graphes en niveaux n'est évidemment plus adaptée dans ce cadre où les graphes dépassent de loin les capacités des supports visuels. Mais d'un point de vue plus théorique, hors du cadre applicatif de la thèse, nous avons voulu en savoir plus sur le comportement des deux métaheuristiques utilisées et le changement de structure de l'espace de recherche.

L'extension de AGH pour le tracé dynamique est présentée dans le paragraphe 5.1 et le paragraphe 5.2 présente les résultats des comparaisons sur les graphes de grandes tailles.

5.1 Tracé dynamique

Comme le note Branke [12], le problème d'ajout de sommets et d'arcs est similaire à celui de leurs suppressions. En effet, la suppression d'un ou plusieurs sommets peut entraîner des "trous" dans la représentation visuelle. Ces espaces sans sommets doivent être comblés en optimisant le tracé et tout en conservant une ressemblance entre les différents tracés. On retrouve alors le problème de tracé dynamique tel que nous l'avons énoncé. Donc, seul le problème d'ajout est traité ici.

Entre deux tracés G_{t-1} et G_t aux instants $t-1$ et t, la carte mentale de l'utilisateur doit être préservée au mieux. Cette préservation est mesurée en ajoutant une contrainte de stabilité à AGH entre les tracés à $t-1$ et t.

La contrainte de stabilité est mesurée en calculant le critère de similarité $\delta(\Pi_t, \Pi_{t-1})$, suggéré par Bridgeman et Tamassia [14], entre les tracés Π_t de G_t à l'instant t et Π_{t-1} de G_{t-1} à l'instant $t-1$. Il est basé sur le nombre de paires de sommets inversées entre les deux tracés :

$$\delta(\Pi_t, \Pi_{t-1}) = 1 - \frac{1}{h} \times \sum_{k=1}^{h} \frac{C_k(t-1,t)}{P_k(t-1,t)} \tag{5.1}$$

où h est le nombre de niveaux du graphe, $P_k(t-1,t)$ le nombre de paires de sommets communes aux deux tracés et $C_k(t-1,t)$ le nombre de paires de sommets inversées entre le tracé à $t-1$ et celui à t. Si $\delta(\Pi_t, \Pi_{t-1})$ est proche de 1 alors le nombre de sommets inversés est petit et le tracé que l'on avait à $t-1$ est par conséquent conservé à l'instant t.

5.1.1 Intégration à AGH

L'algorithme 5.1 décrit la procédure AGH_D qui est l'intégration de la contrainte de stabilité à AGH. Dès que le graphe atteint un ordre suffisant, la contrainte de stabilité est prise en compte uniquement sur les meilleurs tracés de la population finale de AGH. L'algorithme réalise ainsi un bon compromis entre la réduction du nombre de croisements d'arcs et la conservation de la carte mentale du tracé à $t-1$.

La première boucle "Tant que" est utilisée quand le graphe à dessiner est petit ($n_t < 20$). La contrainte dynamique n'est pas considérée et l'on ne retient simplement que la meilleure solution renvoyée par AGH. En effet, les changements apportés au tracé par l'ajout de sommets et d'arcs sont trop importants et l'on ne peut pas considérer qu'une structure stable à préserver existe déjà.

Dès que l'ordre du graphe est suffisant, la contrainte de stabilité n'est prise en compte que pour la population finale de AGH. Plus précisément, AGH est appliqué sur Π_t puis on ne conserve que les meilleurs individus représentant le quart de la population finale. Le critère de stabilité est ensuite calculé sur ces derniers. Le tracé choisi est celui qui en priorité maximise $\delta\left(\Pi_t, \widehat{\Pi_{t-1}}\right)$. Si plusieurs individus sont associés à la même valeur de δ, on ne conserve que ceux qui minimisent le nombre de croisements ; ou on choisit aléatoirement en cas d'égalité sur le nombre de croisements. Le fait de ne conserver que les meilleurs individus de AGH garantit que l'algorithme ne produit pas un

Algorithme 5.1 Procédure AGH_D pour le tracé dynamique

Entrée : Un graphe G_t et un tracé Π_t associé
Sortie : Tracé $\widehat{\Pi_t}$ de G_t intelligible visuellement proche de celui obtenu à $t-1$ pour Π_{t-1}
Variables :
meilleures_solutions : variable de type tableau qui contient des individus de AGH
t : variable qui représente les différents moments du tracé.

Début
$t \leftarrow 0$
Tant Que $n_t < 20$ **faire**
Appliquer AGH sur Π_t
Afficher le meilleur tracé $\widehat{\Pi_t}$
$t \leftarrow t+1$
Fin Tant que
Tant Que critère de fin non atteint **faire**
Appliquer AGH sur Π_t
Mettre dans *meilleures_solutions* les meilleurs individus
de la population finale de AGH
Pour chaque tracé Π_t^i de *meilleures_solutions* **faire**
calculer $\delta\left(\Pi_t^i, \widehat{\Pi_{t-1}}\right)$
Fin pour
Sélectionner le tracé $\widehat{\Pi_t}$ qui minimise $c\left(\Pi_t^i\right)$ et maximise $\delta\left(\Pi_t^i, \widehat{\Pi_{t-1}}\right)$
dans *meilleures_solutions*
Afficher $\widehat{\Pi_t}$
$t \leftarrow t+1$
Fin Tant que
Fin

Chaque modification de la variable t entraîne des ajouts de sommets et d'arcs au graphe G_t (algorithme 5.2)

tracé qui possède une similarité forte avec le tracé précédent et un nombre de croisements important.

5.1.2 Générateur de tracés dynamiques

A la différence du tracé statique, il n'y a pas à notre connaissance de bases de tests disponibles pour des expérimentations à grande échelle pour du tracé dynamique. Bien souvent, les expérimentations sont menées avec des sujets humains auxquels on demande de choisir une solution parmi un ensemble de réponses possibles. Cette approche est bien évidemment nécessaire pour valider la méthode, mais plusieurs problèmes se posent : disponibilité des sujets, qualité subjective des conclusions dues aux différences de sensibilités, et limitation forte du nombre de tracés qui peuvent être évalués. Pour obtenir des mesures quantitatives et valider l'approche sur un grand nombre de graphes, nous avons développé un générateur automatique qui simule un tracé dynamique aléatoire. Ce générateur est appelé par AGH_D pour simuler les différentes étapes du processus de construction du graphe. Ces étapes sont symbolisées ici par les modifications de la variable t. Le principe de fonctionnement du générateur est présenté dans l'algorithme 5.2.

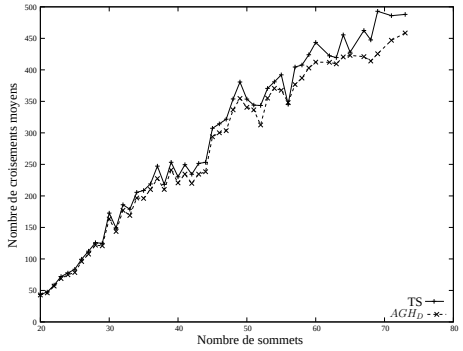
La phase d'initialisation qui engendre un petit graphe aléatoire connexe permet de simuler le début de toute phase de construction où l'ordre du graphe à construire est faible. Ensuite, à chaque instant t, dès qu'une solution a été calculée par AGH_D pour G_t, un nouveau graphe G_{t+1} est engendré en ajoutant aléatoirement des sommets à G_t. Ils peuvent être placés dans les niveaux existants ou dans un nouveau niveau. A chaque fois, on s'assure de ne conserver qu'une seule composante connexe au graphe en ajoutant au moins un arc entre le nouveau sommet et le reste du graphe. Une densité convenable est conservée en ajoutant des arcs supplémentaires, placés aléatoirement.

Le processus de génération s'arrête dès que les paramètres finaux du graphe sont atteints. Pour conserver des graphes pouvant être manipulés par des utilisateurs sur un support standard, la procédure de tracé vérifie quelques autres caractéristiques : le nombre maximal de sommets est ainsi fixé à 80 ($nb\_max\_sommets$ dans l'algorithme 5.2), le nombre maximal d'arcs est fixé à 400 (nb_arcs_max), le nombre maximal de croisements ne doit pas dépasser 500 ($nb\_croisements\_max$) après l'optimisation du tracé. Si une de ces caractéristiques dépasse le seuil fixé, la procédure de tracé s'arrête. De plus le nombre maximal de niveaux est fixé à 15 ($nb\_niveaux\_max$) et la densité des graphes engendrés doit être comprise entre 0.3 ($densite\_min$) et 0.7 ($densite\_max$). Les nouveaux sommets peuvent être placés dans les niveaux existants ou dans un nouveau niveau si le nombre maximum n'est pas encore atteint. Le nombre maximal de sommets pouvant être ajoutés à

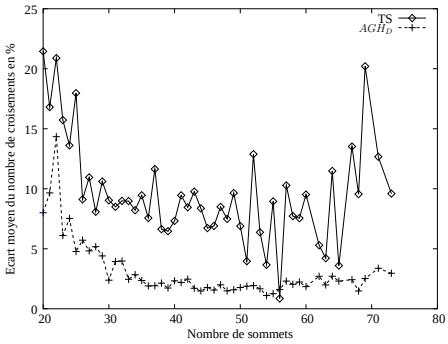
Algorithme 5.2 Générateur de graphes aléatoires pour simuler un tracé dynamique.

Sortie : Un graphe G_t

Début
$nb\_max\_sommets \leftarrow 80$
$nb\_croisements\_max \leftarrow 500$
$nb\_arcs\_max \leftarrow 400$
$nb\_niveaux\_max \leftarrow 15$
$nb\_max\_sommets\_a\_ajouter \leftarrow 2$
$densite\_min \leftarrow 0.3$
$densite\_max \leftarrow 0.7$
Si $t == 0$ *alors*
  Définir les paramètres du graphe G final
  Engendrer un petit graphe G_t connexe avec moins de 20 sommets
Sinon
  Ajouter des sommets à G_{t-1} pour créer un graphe G_t connexe
  Ajouter des arcs pour conserver $0.3 \leq d(G_t) \leq 0.7$
Fin Si
Renvoyer G_t
Fin



(a) Nombre moyen de croisements pour la solution retenue par AGH_D et celle retenue par TS.



(b) Ecart moyen du nombre de croisements des solutions de AGH_D et TS avec la meilleure solution AGH.

FIG. 5.1 – Analyse de la qualité des tracés.

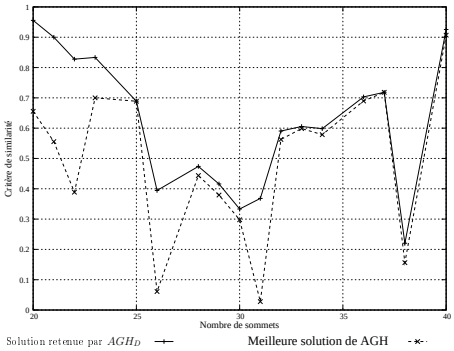
chaque phase est fixé à 2. Des expérimentations avec un nombre plus importants montrent un arrêt rapide de la procédure de dessin puisque les caractéristiques maximales du graphe sont atteintes plus rapidement. Il n'y a en revanche aucune différence de comportement entre les différentes versions.

5.1.3 Résultats expérimentaux

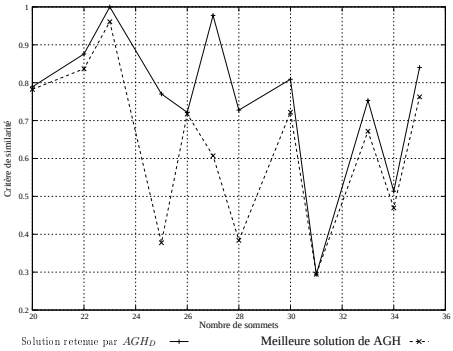
Ce générateur a été utilisé pour simuler le processus de construction de 90 graphes, correspondant à un total de 1400 tracés. La densité moyenne des tracés est $d(G_t) = 0.65$ (écart-type de 0.05).

La figure 5.1-a montre l'évolution du nombre moyen de croisements pour l'ensemble des 90 graphes. L'heuristique basée sur la recherche Tabou (TS) présentée au chapitre 3 paragraphe 3.2.2 page 42 a aussi été utilisée à titre de comparaison. On voit bien ici que AGH_D est toujours meilleur que TS sauf dans un cas. La figure 5.1-b montre, pour TS et la solution retournée par AGH_D, la différence moyenne du nombre de croisements avec la meilleure solution retournée par AGH (tracé statique). Grâce à la construction de l'algorithme, la solution retenue par AGH_D est proche de la meilleure solution en terme de croisements. En moyenne, la solution retenue par AGH_D a 3.14% (écart-type de 2.42%) de croisements en plus que la meilleure solution de AGH. Cette différence est nettement plus importante avec TS : 9.64% (écart-type de 4.28%).

Pour mieux comprendre l'intégration de la contrainte de stabilité, nous avons aussi étudié l'évolution du critère de similarité $\delta(\Pi_t, \Pi_{t-1})$. Ce critère est spécifique à chaque étape, on ne peut donc pas calculer de courbes



(a) Un graphe avec au maximum 40 sommets.



(b) Un graphe avec au maximum 35 sommets.

FIG. 5.2 – Evolution du critère de similarité pour deux graphes représentatifs.

moyennes comme précédemment car cela n'aurait aucune signification. La figure 5.2 montre l'évolution du critère de similarité sur l'ensemble du processus de construction pour deux graphes représentatifs du comportement global. La figure montre à la fois l'évolution de la valeur du critère de similarité pour la solution retenue par AGH et la solution retenue par AGH_D. On peut remarquer que la similarité est nettement meilleure pour AGH_D. Les différences importantes qui peuvent apparaître entre deux étapes de la construction sont dues au processus dynamique : la position des nouveaux sommets et des nouveaux arcs peut parfois conduire à avoir des graphes G_t et G_{t-1} très différents l'un de l'autre. Dans ce cas, les meilleurs individus calculés par AGH contiennent des tracés de G_t très différents de ceux de G_{t-1}. L'algorithme produit alors un nouveau dessin pour respecter la contrainte de lisibilité. Globalement, la stratégie appliquée ici tend à préserver les ressemblances entre les tracés.

5.2 Tracé de grands graphes

D'après les résultats du test de Wilcoxon effectué au chapitre 4 paragraphe 4.5.2 page 83, la différence de performance entre AGH et la méthode de descentes DMb pour les graphes les plus denses avec un nombre important de niveaux n'est plus significative. Nous avons voulu vérifier si ce changement de comportement des deux métaheuristiques, obtenu sur un nombre restreint d'instances, se confirme sur un ensemble plus conséquent. Nous avons donc comparé à nouveau AGH et DMb sur un ensemble plus important de graphes de grandes tailles.

5.2.1 Paramètres expérimentaux

Le générateur de graphes du chapitre 4 a été utilisé pour engendrer une nouvelle famille de 180 graphes comportant l'ensemble des combinaisons possibles de 20, 25 ou 30 niveaux pour $d(G) = 0.6$, $d(G) = 0.65$ ou $d(G) = 0.7$. Le nombre de sommets par niveau est compris entre 10 et 35. Pour donner un ordre de grandeur, le nombre moyen de croisements est ici de l'ordre de 485 000 sommets (écart-type de l'ordre de 170 000).

Nous utilisons le même processus expérimental que pour les comparaisons du chapitre précédent : AGH est exécuté en premier et DMb a ensuite un temps de calcul égal au temps mis par AGH pour trouver la meilleure solution. Puisque le temps de calcul n'est plus une contrainte, pour obtenir les meilleures solutions possibles la condition d'arrêt de AGH est d'effectuer 500 générations sans amélioration du meilleur résultat au lieu des 100 générations effectuées précédemment. Chaque méthode est exécutée 100 fois sur chacun des graphes (total de 18000 exécutions des algorithmes). Pour information, ces nouvelles comparaisons ont nécessité environ 8 mois de calculs au serveur dédié à cette tâche[1].

5.2.2 Résultats expérimentaux

De façon analogue aux comparaisons effectuées dans le chapitre précédent, nous avons calculé le pourcentage moyen de meilleures solutions trouvées (tableau 5.1). Le test de Wilcoxon effectué sur chaque combinaison des paramètres (9 combinaisons de 2000 exécutions de chaque algorithme) confirme dans tous les cas que les descentes multiples sont meilleures que AGH.

Pour ces graphes de grandes tailles où le nombre d'optima locaux est important et leur qualité parfois très proche, il semble que le paysage de recherche ne possède pas de structure permettant à AGH d'utiliser ses capacités intrinsèques d'apprentissage. Une exploration à large spectre du paysage de recherche effectuée par une méthode simple et rapide telle que les descentes multiples permet dans ce cas de trouver une meilleure solution.

5.3 Conclusion

Puisque les temps de calculs de AGH sont compatibles avec du tracé dynamique, aspect important de tout processus de construction de graphes, nous avons décidé, suite à des premières expérimentations réduites [86], de

[1]Bi-processeur AMD Athlon MP 2400+. Les processeurs fonctionnent à 2GHz. Ces calculs représentaient la tâche principale du serveur.

h	20	25	30	Moy.
AGH	0	0	0	0
DMb	100	100	100	100
AGH=DMb	0	0	0	0

a. Résultats pour $d(G) = 0.6$

h	20	25	30	Moy.
AGH	0.05	0	0	0.02
DMb	99.95	100	100	99.98
AGH=DMb	0	0	0	0

b. Résultats pour $d(G) = 0.65$

h	20	25	30	Moy.
AGH	0.05	0	0	0.02
DMb	99.95	100	100	99.98
AGH=DMb	0	0	0	0

c. Résultats pour $d(G) = 0.7$

La colonne AGH=DMb signifie que les deux approches ont trouvé une solution avec le même nombre de croisements.

TAB. 5.1 – Comparaisons du % moyen de meilleures solutions trouvées entre AGH et les descentes multiples (DMb).

l'étendre à ce problème. L'ajout d'une contrainte de similarité pour respecter au mieux la carte mentale de l'utilisateur permet bien souvent de préserver les ressemblances entre deux tracés aux instants $t-1$ et t du processus de construction tout en ne détériorant pas trop le nombre de croisements d'arcs. Quand cette préservation n'est pas possible, lorsque les graphes à $t-1$ et t sont trop différents, l'algorithme privilégie plutôt un tracé lisible et compréhensible au détriment d'un effort cognitif plus important à fournir par l'utilisateur.

En dehors du cadre applicatif de la thèse, pour approfondir d'un point de vue plus théorique l'analyse du comportement de AGH et DMb, nous avons comparé à nouveau ces deux méthodes sur un ensemble de graphes de grandes tailles ; ce qui a nécessité des calculs importants (plus de 8 mois sur un bi-processeur AMD Athlon MP 2400+). Les résultats montrent que les descentes surclassent nettement l'algorithme génétique. Ces résultats suggèrent un changement important de la structure de l'espace de recherche lors du passage à l'échelle.

Chapitre 6

Application sur Atanor

Sommaire

Le système de gestion des connaissances Atanor qui sert ici d'environnement applicatif de référence est une mémoire d'entreprise procédurale qui permet aux experts de capitaliser leurs connaissances et de les relier aux différents éléments du système d'information de l'entreprise (GED, workflow, organigramme, ...). Ces connaissances sont ensuite restituées aux utilisateurs pour être acquises dans une phase d'aide à la décision. Pour les besoins de cette thèse, nous nous restreignons au domaine d'application de l'aide à la maintenance dans un contexte industriel. Dans ce cadre, Atanor permet de réduire les temps d'arrêt d'une machine lors d'un défaut ou d'une panne en aidant l'utilisateur à localiser les composants défectueux ou le problème en fonction des symptômes présents. En particulier, nous utilisons les modèles de connaissances définis pour l'aide à la maintenance d'un type de machines de tri automatique de courrier de la Poste. Il existe bien d'autres

exemples concrets d'utilisation d'Atanor : dans le domaine industriel on peut notamment citer l'aide à l'uniformisation de procédures dans le cadre de la conservation et de la collecte du lait ; l'aide à l'accélération de la reconfiguration de chaînes de production dans la cuisine industrielle ; l'aide à la gestion de crise et à la traçabilité des produits dans une centrale d'achat ; dans le domaine militaire on peut citer l'aide au maintien en fonctionnement de sous-marins ou de navires de surface. Des applications dans le secteur des services sont aussi possibles comme dans les centres d'appels pour venir en aide aux téléconseillers.

Lors de la phase de capitalisation des connaissances des experts qui nous intéresse ici, pour des impressions *a priori* de simplicité, les modèles d'arbres ont souvent été privilégiés en GC. Le modèle visuel utilisé initialement dans Atanor pour réaliser l'interfaçage entre le serveur de connaissances et l'expert peut être vu comme une généralisation enrichie du modèle des arbres de défaillance et des arbres de décision ; il est appelé *logigramme d'expertise*. Les différents retours d'expériences, dans différents contextes applicatifs ont montré que cette structuration en logigramme n'est finalement pas la plus intuitive pour les experts car de nombreuses redondances de sommets ou de sous-arbres sont présentes. Ces redondances ne sont pas dues à un défaut de conception du serveur mais à la nature des processus à traiter. Des procédures particulières de construction ainsi que de longues formations qui nécessitent un effort cognitif et un investissement personnel important de la part des experts sont nécessaires. Les premières utilisations sont souvent déstabilisantes car l'expert doit se conformer au mode de fonctionnement de l'outil et non le contraire.

En revanche, ce problème de redondances est déjà résolu pour le stockage interne des connaissances. En effet, les bases de connaissances sont structurées sous la forme de graphes dans lesquels chaque sommet n'est décrit qu'une seule fois. Cette structuration en graphe est de loin la plus efficace. C'est la transformation de ces graphes en arbres qui introduit les redondances.

Pour obtenir un outil qui soit plus compatible avec le mode de raisonnement utilisé par les experts et ainsi éviter cette fastidieuse transformation, nous avons proposé le modèle Graph'Atanor. Ce modèle utilise des graphes en niveaux pour représenter les processus. Ce type de graphe est simple à utiliser : les sommets représentent les différentes tâches qui sont ici les différentes étapes de résolution d'une panne et les niveaux symbolisent l'enchaînement de ces étapes. Ce modèle reprend les différentes caractéristiques du modèle d'arbres tout en apportant une lisibilité accrue et de meilleures possibilités d'exploitation des modèles de connaissances. L'algorithme AGH est utilisé pour réaliser les représentations visuelles de Graph'Atanor.

Le premier paragraphe de ce chapitre présente un descriptif général d'Ata-

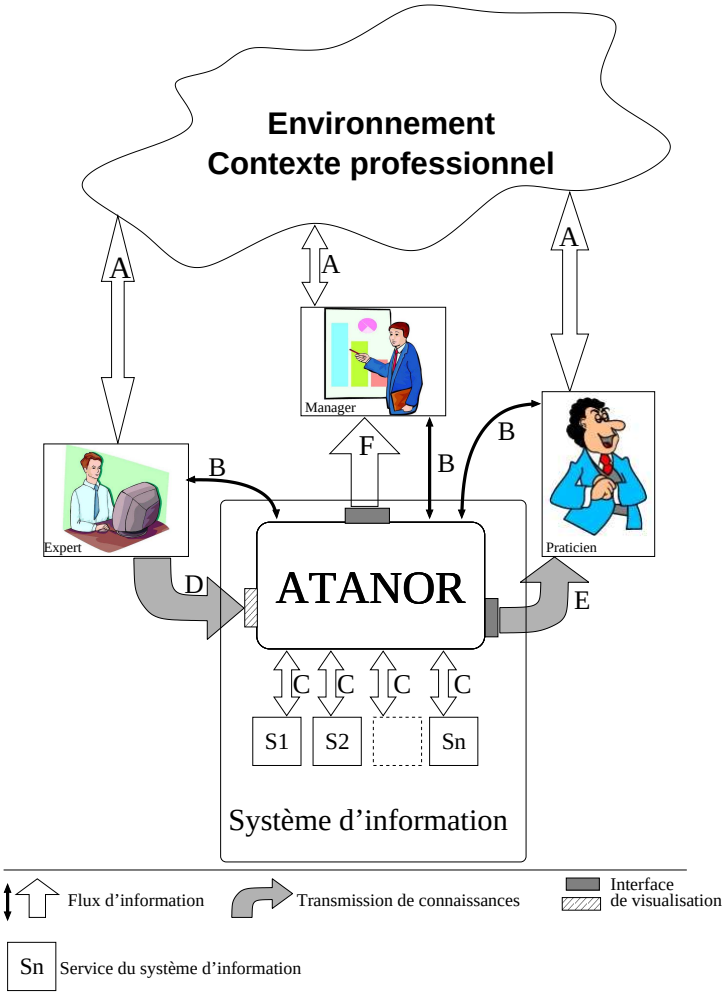
nor, de ses modèles de connaissances et des différents modules : *Expert* pour construire et faire évoluer les modèles, *Praticien* pour exploiter les connaissances et *Manager* pour les évaluer. Le paragraphe 6.2 présente le modèle des logigrammes qui sert de support à la représentation visuelle des connaissances à destination des experts. Le paragraphe 6.3 introduit le nouveau modèle Graph'Atanor. Plusieurs comparaisons sont effectuées dans le paragraphe 6.4 pour montrer les différents apports de Graph'Atanor par rapport aux logigrammes.

6.1 Descriptif général d'Atanor

Atanor occupe une position centrale dans le système d'information de l'entreprise (figure 6.1). Il permet de gérer l'ensemble des étapes d'un processus de gestion des connaissances (voir la figure 1.1 page 3) :

1. au début du processus de GC, les flèches notées A représentent la détection des besoins ;
2. la flèche notée D symbolise la phase de construction des modèles de connaissances ;
3. la diffusion des connaissances et l'utilisation du système sont représentées par la flèche notée E et dans une moindre mesure par les différentes flèches notées B ;
4. les flèches notées A et F ainsi que les différentes flèches notées B représentent l'évaluation du système. Pour ces dernières l'utilisateur peut, par exemple, envoyer un message à l'expert pour lui signaler une erreur, l'expert peut proposer des modifications au manager avant de les faire, ...
5. La maintenance et l'évolution sont représentées par la flèche notée D.

Chaque interface de visualisation des connaissances représente un module du serveur de connaissances. Ces trois modules communiquent de façon séparée avec le noyau du serveur de connaissances écrit en Prolog. Le choix de ce langage facilite la gestion interne des connaissances recueillies, mais surtout permet de les activer. Il offre également de grandes perspectives d'extension du modèle de connaissances actuel orienté diagnostic, afin de traiter d'autres type de tâches et d'autres formes de connaissances actionnables (inférences).



- *Flèches A : l'utilisateur, l'expert et le manager prennent des informations dans leur environnement de travail pour effectuer leurs tâches. Ils sont ainsi en interaction permanente avec cet environnement grâce à Atanor.*
- *Flèches B : les utilisateurs, les experts et les managers peuvent échanger des messages qui portent sur les connaissances maintenus par Atanor.*
- *Flèches C : Atanor peut interagir avec les autres services du système d'information de l'entreprise (GED, bases de données, serveur de mails, ...).*
- *Flèche D : l'expert capitalise ses connaissances dans Atanor. Cette étape s'effectue au moyen d'une interface de visualisation spécifique.*
- *Flèche E : l'utilisateur acquiert des nouvelles connaissances en utilisant Atanor par l'intermédiaire d'une interface de visualisation adaptée.*
- *Flèche F : une interface de visualisation spécifique permet d'informer le manager sur la façon dont le serveur est utilisé par les experts et les utilisateurs.*

Le travail de cette thèse se situe au niveau de l'interface de visualisation à destination des experts (le rectangle hachuré).

FIG. 6.1 – Positionnement d'Atanor dans l'entreprise

6.1.1 L'éditeur graphique des connaissances : le module *Expert*

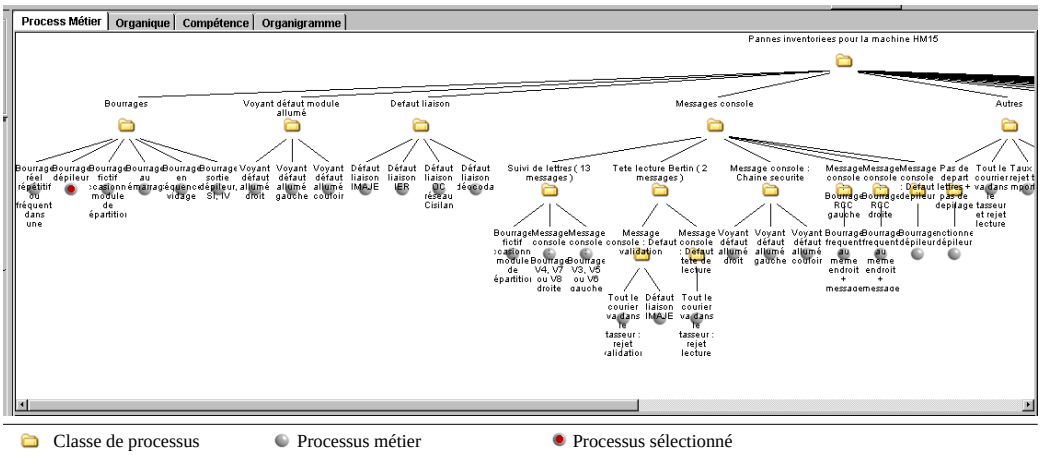
Cet éditeur graphique est utilisé pour permettre aux experts de créer, modifier et faire évoluer les différents modèles composant la base de connaissances. Les interfaces spécifiques interactives de chaque modèle sont accessibles au moyen d'onglets présents en haut de la fenêtre (voir figure 6.2-a).

En particulier, l'interface du modèle expertise est décomposée en deux. La partie haute est utilisée pour représenter la hiérarchie des différents processus métiers relatifs au système sous la forme d'un arbre (figure 6.2-a). Les sommets intermédiaires représentent des classes de processus et chaque feuille représente dans le cadre de notre exemple une panne possible de la machine de tri. La partie basse de l'interface permet de visualiser la description détaillée d'un processus sélectionné dans la hiérarchie, sous la forme d'un logigramme (figure 6.2-b).

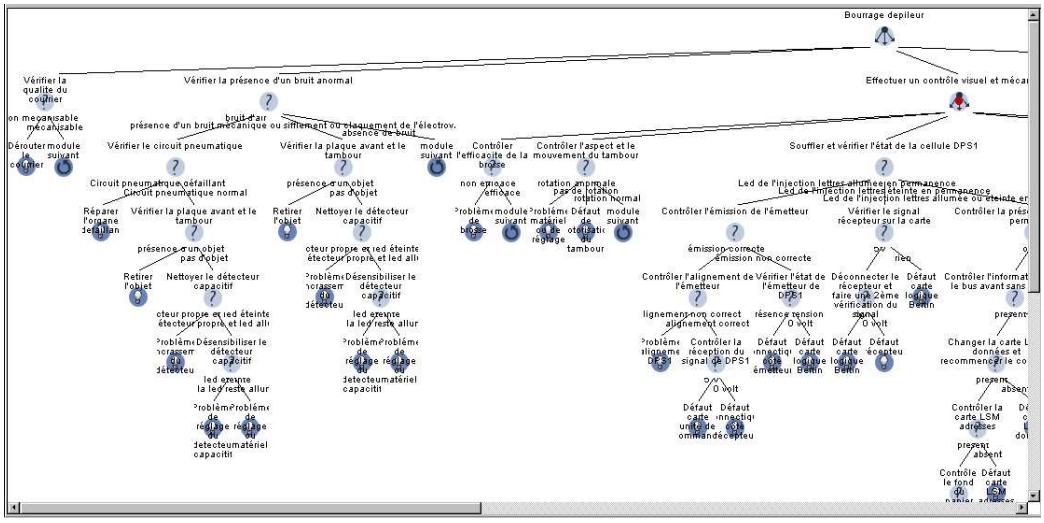
Les connaissances décrites par ce formalisme graphique sont ensuite stockées sous forme relationnelle en prédicats Prolog dans une base de connaissances activables. Ces connaissances peuvent ensuite être déclenchées par les utilisateurs du système par l'intermédiaire du module *Praticien*.

6.1.2 L'aide à la décision : le module *Praticien*

Ce module est conçu pour les utilisateurs du système. Il permet de dérouler de façon interactive un questionnaire d'aide à la décision conçu à partir du modèle d'expertise. Le principe de fonctionnement se décompose en deux étapes comme dans le module *Expert*. La hiérarchie des processus métiers est tout d'abord présentée sous la forme d'une arborescence pour permettre le choix d'un processus particulier (figure 6.3). Ensuite, le logigramme qui correspond au processus sélectionné est présenté à l'utilisateur sous la forme d'un questionnaire interactif (figure 6.4). Les réponses données permettent de parcourir le logigramme. A chaque étape du questionnaire, l'utilisateur retrouve les différents éléments saisis par l'expert tels qu'une description du *mode opératoire* à suivre permettant par exemple de vérifier l'état d'un composant, des associations avec des éléments multimédias (sons, vidéos, images, documents divers), des liaisons vers les autres modèles de connaissances, une liaison vers une documentation complémentaire (GED) ou bien encore des modèles en réalité virtuelle qui peuvent décrire des pièces de la machine [43]. Le serveur conserve une trace des différentes actions de l'utilisateur dans une base historique pour une exploitation ultérieure par le module *Manager*.



(a) Extrait de la hiérarchie des différentes pannes possibles de la machine de tri.



(b) Extrait du processus sélectionné

FIG. 6.2 – Le module *Expert* d'Atanor.

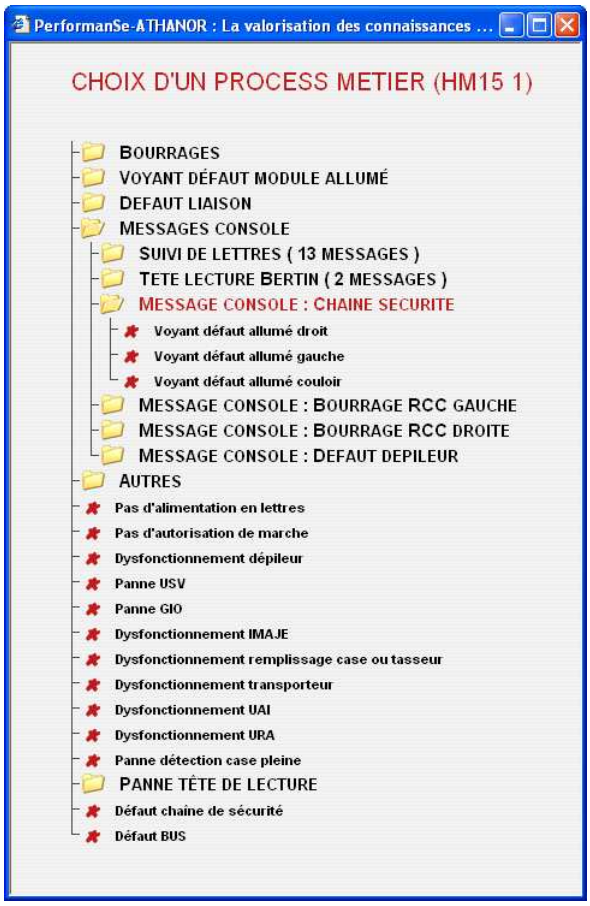
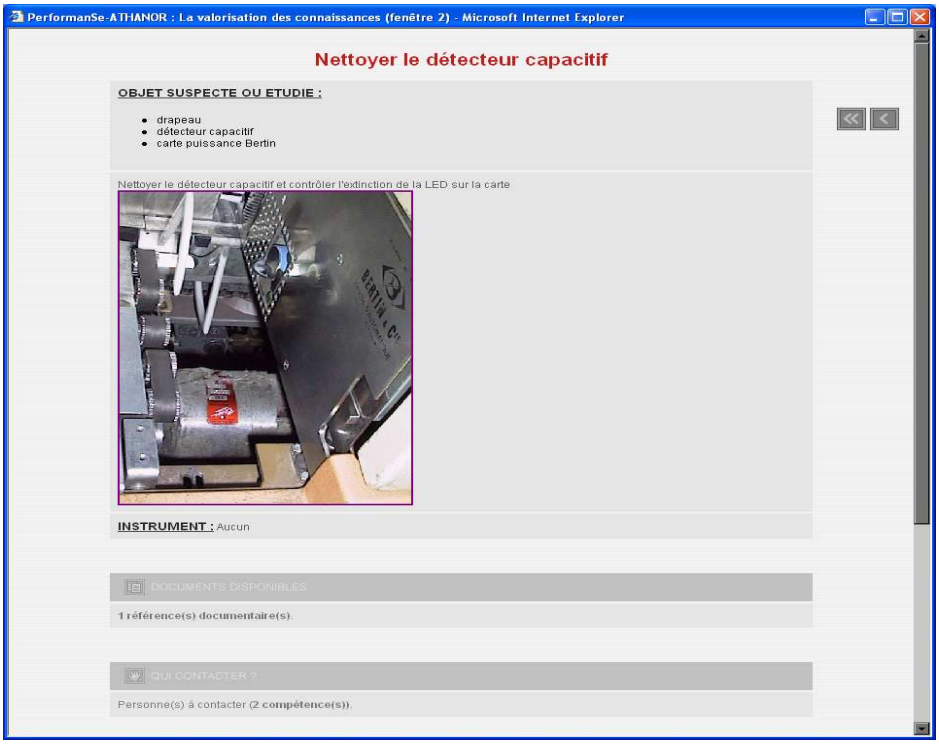
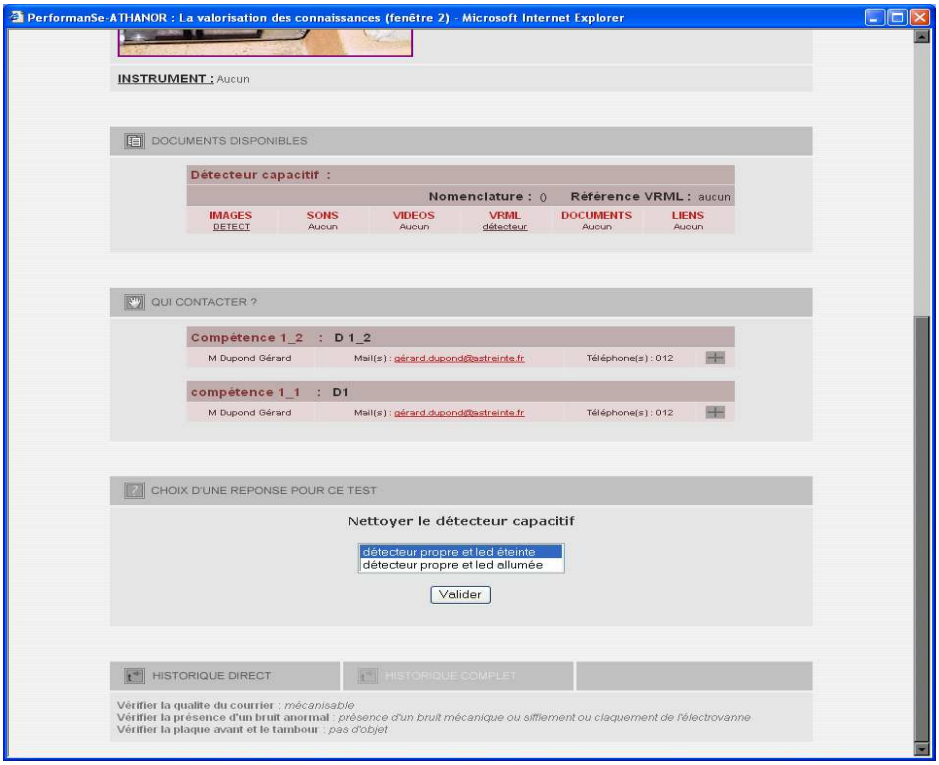


FIG. 6.3 – Module *Praticien* : choix d'une panne.



(a) Le haut de la page



(b) Le bas de la page

FIG. 6.4 – Module *Praticien* : exemple de test proposé à l'utilisateur.

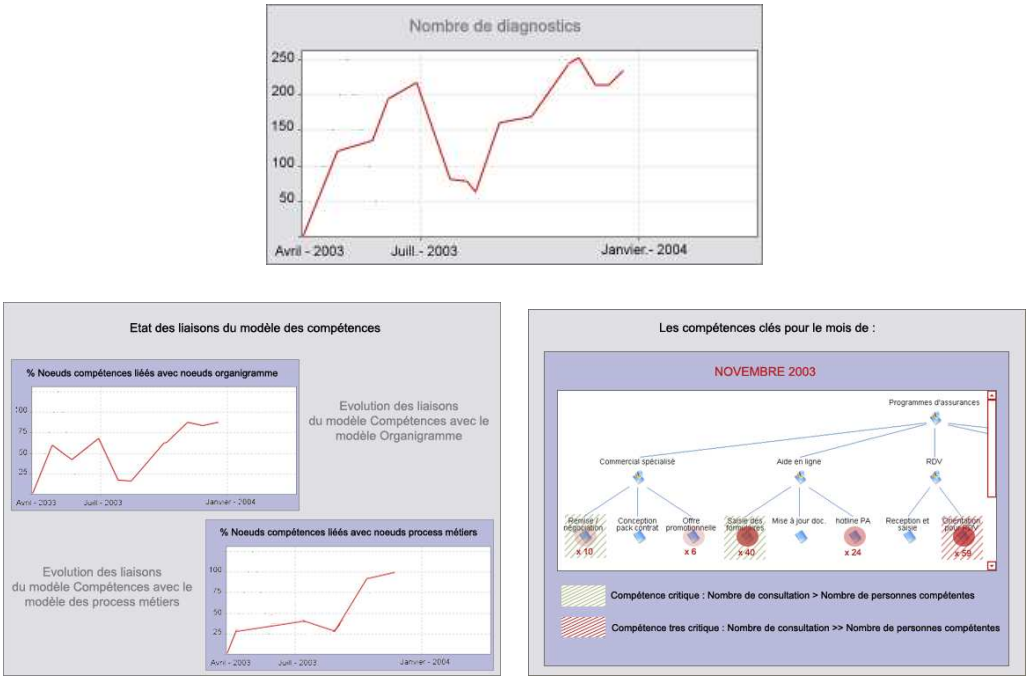


FIG. 6.5 – Différents indices du module *Manager*

6.1.3 Le suivi des connaissances : le module *Manager*

Le module *Manager* est un tableau de bord sur la base de connaissances (figure 6.5). Pour un système complexe, il est important de pouvoir contrôler rapidement l'adéquation du modèle implémenté dans le serveur de connaissances avec le système réel dans son contexte opérationnel. Ce module est une source importante d'informations pour les responsables du système. Des indices structurels, appréhendant la complexité au sens de la modélisation du système, ainsi que des indices mesurant l'utilisation de la base de connaissances sont présentés. Ils permettent d'orienter l'évolution des différents modèles afin notamment de les adapter au mieux au contexte d'exploitation.

6.1.4 Les modèles de connaissances

Les différents modules sont des vues sur les modèles de connaissances. L'approche globale d'Atanor est basée sur quatre modèles :

1. un "modèle expertise" ou modèle des "process métiers" qui permet de représenter les processus métiers en maintenant des connaissances procédurales, exprimées sous formes de règles de raisonnement ;

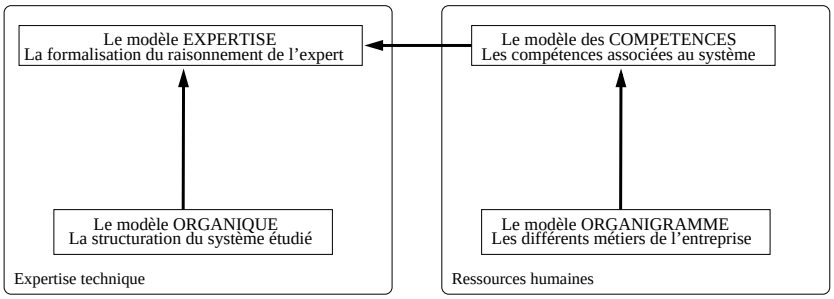


FIG. 6.6 – Les différents modèles d'Atanor et leurs interactions.

2. un "modèle organique" qui décompose le système en un ensemble de sous-systèmes et composants qui peuvent être enrichis par des informations de type multimédia ;
3. un "modèle des compétences" qui permet de décrire le référentiel des compétences des acteurs sur le système (de l'équipe à l'entreprise) en les hiérarchisant du plus global au plus spécifique en terme de savoir, savoir-faire et savoir-être ;
4. un "modèle organigramme" qui contient les personnes qui peuvent intervenir sur le système ou répondre à des questions précises. Cet organigramme peut être associé à un modèle de compétences [134].

Ces modèles interagissent entre eux (figure 6.6) : les connaissances portent sur des composants d'un système dont la manipulation nécessite des compétences elles-même portées par des individus de l'organisation en charge de ce système.

Ces modèles sont directement accessibles par l'intermédiaire de représentations graphiques dans le module *Expert* (figure 6.2). Hormis le modèle expertise qui utilise la représentation en logigramme présentée dans le paragraphe suivant, les autres modèles utilisent des représentations hiérarchiques classiques.

6.2 Le modèle des logigrammes

Ce modèle permet de représenter des connaissances procédurales actionnables liées à un savoir-faire se décomposant en une suite d'étapes. Ainsi, dans le cas de l'aide au diagnostic de pannes, la stratégie mise en œuvre par les experts consiste à tester successivement des hypothèses sur l'état des composants ou des fonctionnalités du système, et ceci en procédant généralement des hypothèses les plus simples aux plus complexes. Ces connaissances

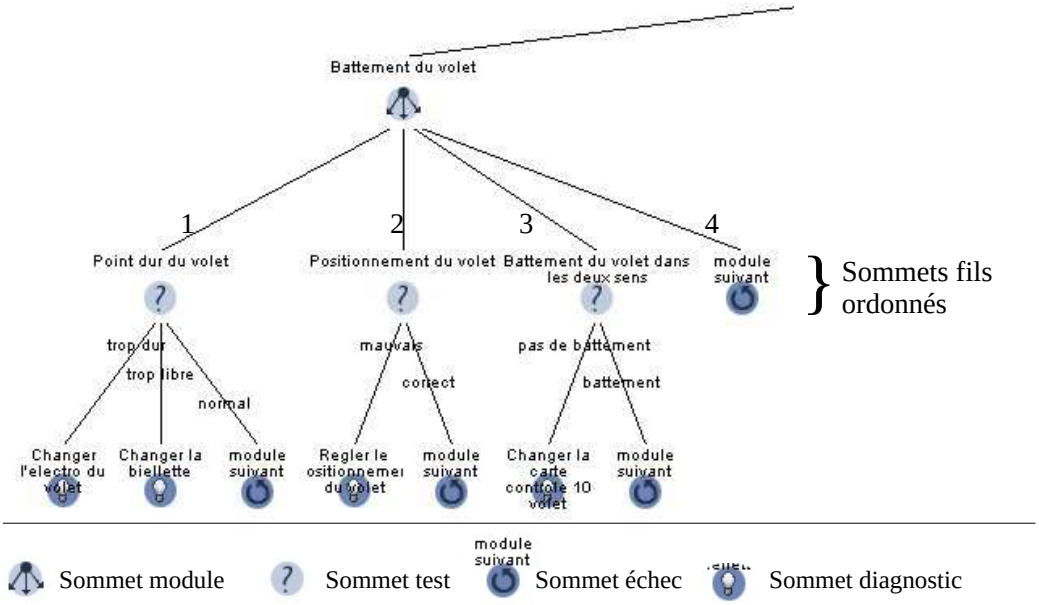


FIG. 6.7 – Formalisation des connaissances par logigramme.

actionnables se traduisent sous la forme d'un arbre appelé *logigramme d'expertise* (Figure 6.7). Le raisonnement de l'expert s'y traduit par un parcours de l'arbre en profondeur d'abord depuis sa racine, chaque étape du raisonnement correspondant à un sommet. Cette représentation graphique peut être vue comme une généralisation des arbres de décision et des arbres de défaillance ; ils sont enrichis de sommets structurants afin de prendre en compte le modèle de raisonnement des experts.

Deux types de sommets structurants sont mis en évidence :

1. Les sommets *modules,* absents des arbres de décision et de défaillance, dont les fils sont ordonnés de gauche à droite et généralement du plus simple au plus complexe, au sens de l'expert. Chacun de ces sommets permet de définir un "module de connaissances" qui permet d'intégrer des principes cognitifs caractéristiques des stratégies de décision expertes [6], dont un principe de parcimonie/décidabilité :
   - le premier sommet fils d'un module permet d'arriver à une décision à moindre coût par des opérations simples (parcimonie),
   - les sommets fils suivants offrent la possibilité de réaliser des opérations de plus en plus complexes afin d'arriver à une prise de décision même si elle s'avère coûteuse (décidabilité).
2. Les sommets *tests* associés à une variable, typiques des arbres de déci-

sion, dont les fils ne sont pas ordonnés, mais dont chaque arc est associé à une valeur de la variable. Dans notre exemple de l'aide au diagnostic, ces valeurs qui servent de transition avec les sommets suivants sont généralement associées à l'état de fonctionnement dans lequel se trouve un élément du système sous-jacent.

Il existe aussi deux types de sommets terminaux : les sommets associés à un *diagnostic* indiquant une solution au problème traité et les actions à effectuer ainsi que les sommets associés à un *échec* indiquant une non résolution du problème. Ces derniers provoquent la mise en œuvre d'un mécanisme de retour au dernier sommet module traité et la transition au sommet suivant au sens de l'ordre induit par ce sommet module.

Une propriété importante de cette formalisation graphique réside dans la possibilité de transformer un logigramme en un ensemble de règles de production, en traduisant l'ensemble des chemins menant de la racine à chacune des feuilles. Ainsi le logigramme de la figure 6.7 se transforme en 4 règles :

Règle 1 : *si* point dur volet = trop dur *alors* changer l'électro du volet ;

Règle 2 : *si* point dur volet = trop libre *alors* changer la biellette ;

Règle 3 : *si* point dur volet = normal *et* positionnement du volet = mauvais *alors* régler le positionnement du volet ;

Règle 4 : *si* point dur volet = normal *et* positionnement du volet = correct *et* battement du volet dans les deux sens = pas de battement *alors* changer la carte contrôle 10 volet ;

La représentation graphique des connaissances par logigramme a l'avantage d'être beaucoup plus intelligible et synthétique qu'un ensemble équivalent de règles de production. Un défaut important est que des sous-arbres correspondant à des sous-ensembles de règles de production utilisés dans différentes phases peuvent être dupliqués à l'issue de la phase d'expertise. Lorsque cette duplication est fréquente, elle peut nuire à l'intelligibilité de la représentation visuelle.

6.3 Le modèle Graph'Atanor

Pour palier aux limites du logigramme d'expertise, nous avons développé un modèle, baptisé Graph'Atanor, basé sur des graphes en niveaux. Il a pour avantage de pouvoir exploiter directement le modèle Prolog du serveur de connaissances qui associe un sommet tel qu'il soit avec l'ensemble de ses fils sans redondance. Les sommets du graphe (figure 6.8) représentent comme dans les logigrammes les tests, les modules, les diagnostics et les

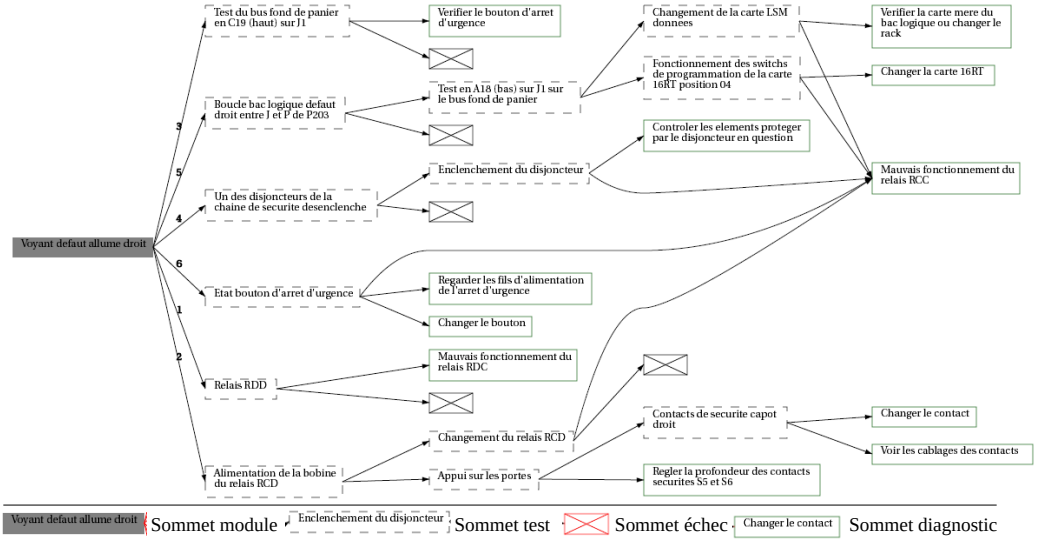


FIG. 6.8 – Représentation d'un processus métier de la machine de tri selon le modèle Graph'Atanor.

échecs. La différence majeure est ici l'unicité ; un sommet ne peut pas être dupliqué. Ils sont ordonnés dans des niveaux verticaux en notant 1 le premier niveau ; le niveau i contient les sommets à distance i –selon le plus court chemin sur le tracé– du niveau 1. Les arcs des sommets tests représentent les différentes valeurs possibles de la variable associée à ce sommet. Pour les modules, comme dans les logigrammes, les arcs sont associés à un numéro d'ordre qui définit la priorité de la transition. Pour obtenir un dessin le plus lisible possible, la contrainte qui impose un ordre des fils dans les logigrammes est relaxée. Le chiffre inscrit au dessus des arcs indique l'ordre dans lequel ils doivent être traités. Pour les sommets tests, les réponses permettant de choisir le chemin à suivre ne sont pas affichées sur les arcs.

Les représentations visuelles respectent au mieux le principe de bonne continuité de la Gestalt [77], illustré au chapitre 2, qui a été validé sur des graphes par Ware *et al.* [137] pour un problème de recherche du plus court chemin. Pour cela, les arcs dont la longueur est supérieure à 1 sont dessinés avec des courbes de béziers quadratiques. Ces courbes particulières permettent d'aplanir les angles et de rendre les tracés plus agréables à regarder.

6.4 Comparaisons expérimentales des modèles

Nous avons comparé les logigrammes et Graph'Atanor dans le cadre applicatif de l'aide au diagnostic pour la maintenance d'un type de machines de tri automatique de courrier de La Poste. Après une période d'adaptation et de formation, les experts des machines de tris ont commencé à assurer la construction, la mise à jour et l'évolution des connaissances maintenues par l'outil. La phase de recueil des connaissances s'est étalée sur deux ans et s'est appuyée sur quatre experts géographiquement dispersés. Elle a permis de mettre en évidence une trentaine de processus métiers, chacun correspondant à une panne possible de la machine, nécessitant la construction d'un logigramme par panne. Les experts ont ainsi fait apparaître plus de 400 sommets tests et environ 200 diagnostics différents ont été répertoriés. Nous avons appliqué le modèle Graph'Atanor sur la hiérarchie des différents processus métiers et sur les logigrammes qui correspondent aux pannes.

6.4.1 Avec la hiérarchie des processus métiers

La figure 6.9 représente les différents processus métiers (dont un extrait est présenté sur figure 6.2-a page 106) en utilisant le modèle Graph'Atanor. On voit clairement que cette hiérarchie est structurée sous forme d'un graphe, les experts travaillant implicitement avec de telles représentations. Selon les experts, certaines pannes peuvent faire partie de plusieurs classes ("Panne tête de lecture Bertin" dans le dernier niveau par exemple), ce qui signifie qu'une panne donnée peut avoir de multiples conséquences dans la machine comme par exemple plusieurs voyants d'erreurs allumés.

6.4.2 Avec les processus métiers

Les figures 6.10 et 6.12-a sont deux extraits de deux logigrammes réalisés avec Atanor. Sans les flèches qui ont été rajoutées sur la première figure, il est difficile au premier abord de constater que plusieurs effets peuvent permettre d'arriver à un même diagnostic qui est dupliqué dans la représentation visuelle. Les sommets modules des logigrammes étant parcourus de la gauche vers la droite, le diagnostic pointé par la flèche la plus à gauche est relativement facile et rapide à réaliser. Le même diagnostic pointé par la flèche la plus à droite est nettement plus difficile et plus long à effectuer car l'utilisateur devra au préalable passer par toutes les branches intermédiaires du sommet module.

La figure 6.12-a illustre ce même problème de duplication mais cette fois pour des sous-arbres du logigramme. Un même diagnostic peut dans ce cas

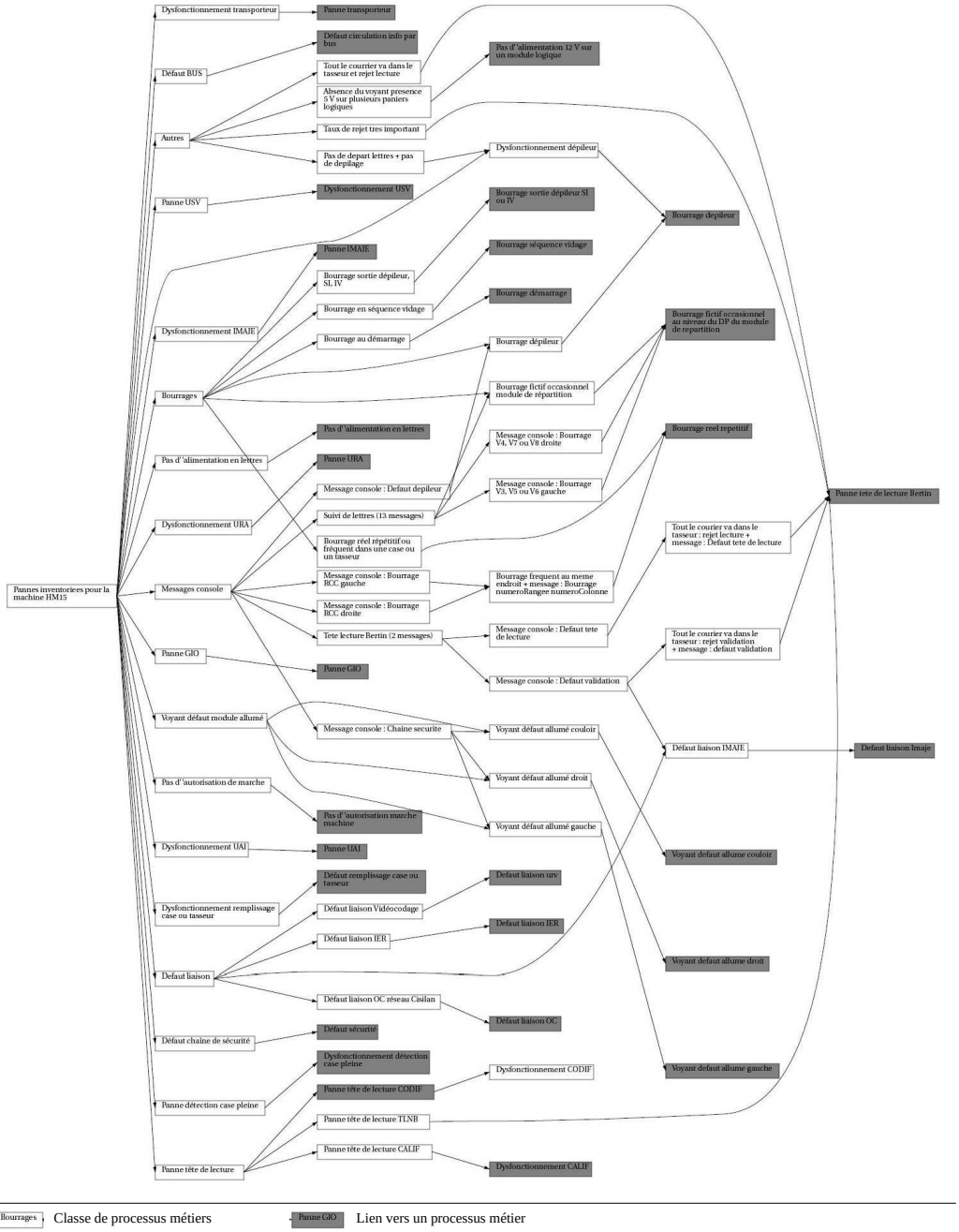
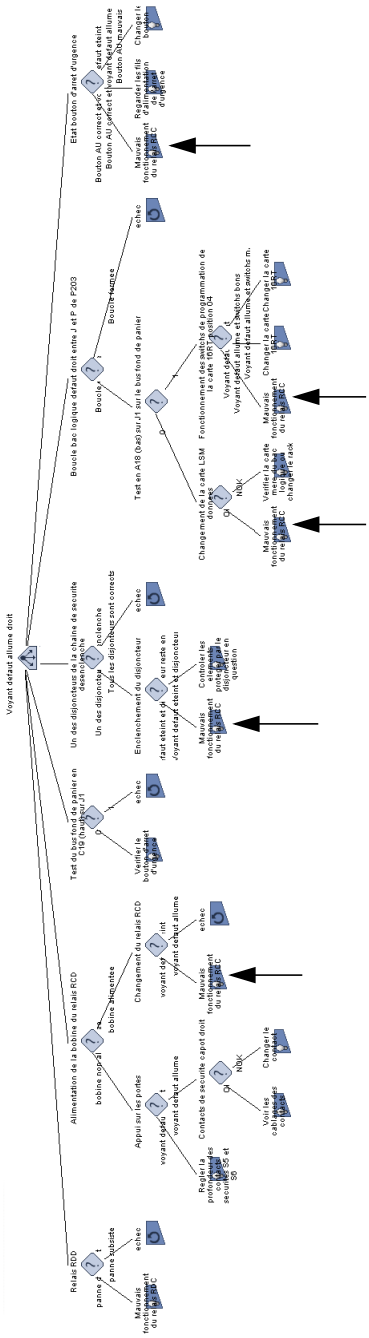
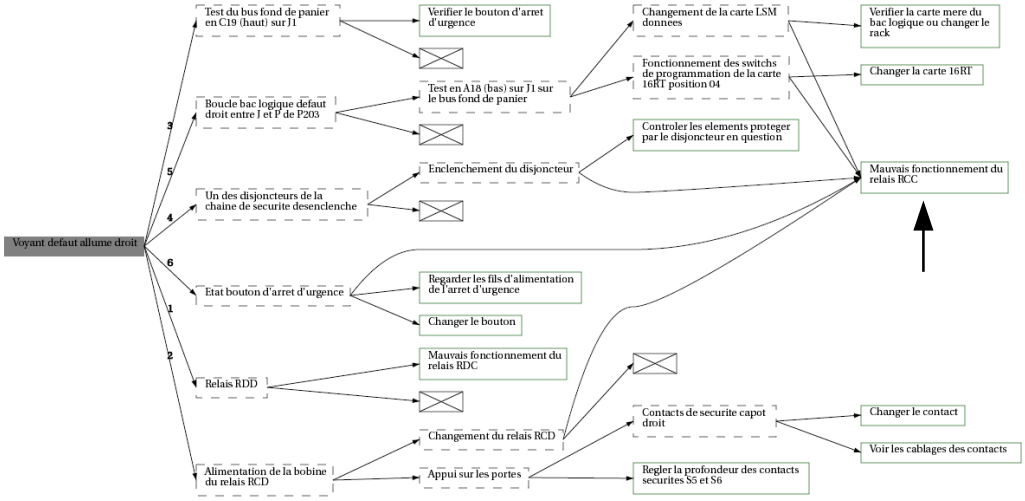


FIG. 6.9 – La hiérarchie des différents processus métiers de la machine de tri de courrier représentée avec le modèle Graph'Atanor.



Les flèches indiquent un même sommet dupliqué.

FIG. 6.10 – Illustration de la duplication de sommets. Extrait du logigramme "Voyant défaut allumé droit" effectué avec Atanor.



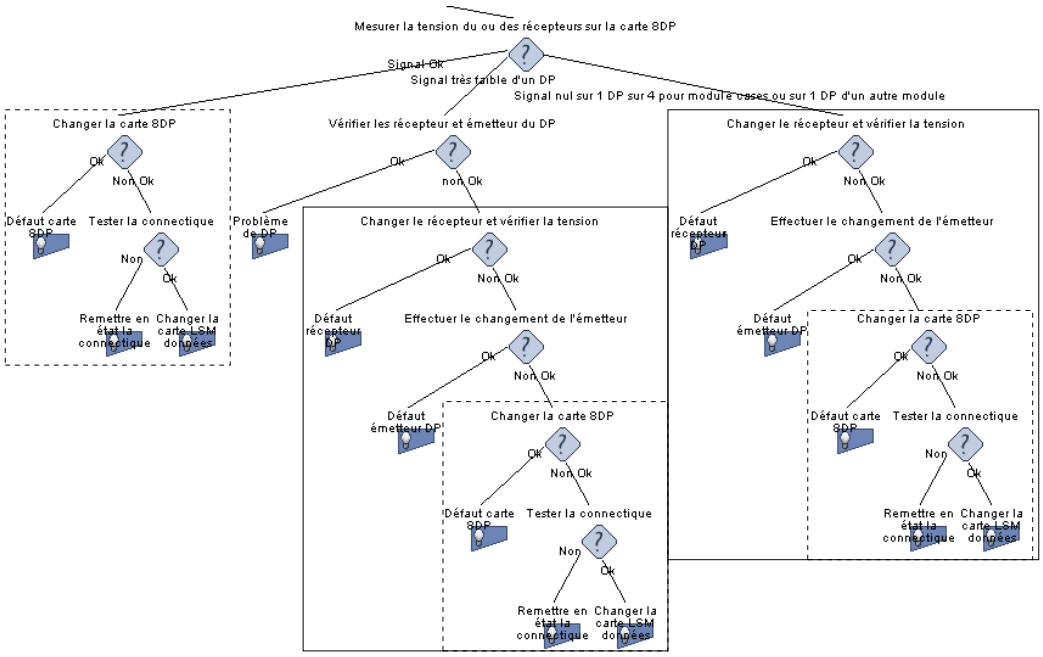
La flèche indique le sommet qui était dupliqué dans la figure 6.10.

FIG. 6.11 – La représentation avec un graphe en niveaux du logigramme de la figure 6.10 pour le processus “Voyant défaut allumé droit”.

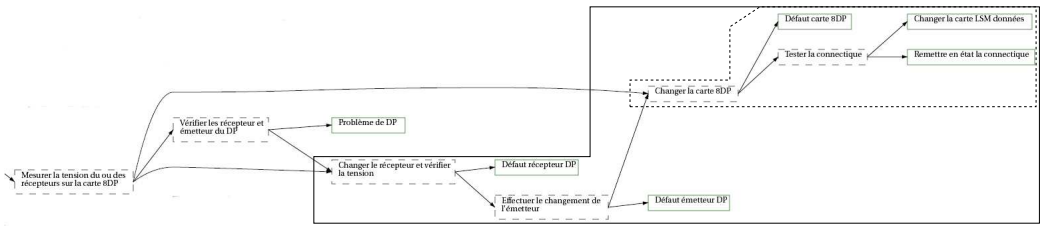
être effectué soit rapidement et simplement, trois étapes intermédiaires sont nécessaires avant d'arriver sur le premier sommet du rectangle en pointillé le plus à gauche, soit de façon plus longue et compliquée : six étapes intermédiaires sont nécessaires pour le rectangle en pointillé du milieu et cinq pour celui le plus à droite. Les figures 6.11 et 6.12-b sont les représentations de ces processus métiers en utilisant le modèle Graph'Atanor.

Ces représentations permettent une meilleure exploitation de la représentation visuelle des modèles de connaissances par l'expert ou le manager grâce notamment à la non duplication des sommets. De plus, les sommets avec un degré important ont statistiquement plus de chances d'être utilisés dans les diagnostics. Utilisés conjointement avec l'historique des différentes actions des utilisateurs ou encore le comptage des passages dans chaque sommet différent (rendu possible grâce à Graph'Atanor), ces différents indices permettent de maintenir une attention particulière sur le système et contribuent ainsi à la maintenance préventive. Une conséquence immédiate est de pouvoir améliorer la répartition des experts ou des techniciens pour être certain d'avoir toujours une personne compétente présente en cas de panne risquant se produire souvent. Ces statistiques permettent à l'expert ou au manager de recenser les composants peu utilisés du graphe (pannes peu fréquentes) en les distinguant de ceux qui le sont fréquemment (pannes fréquentes pouvant indiquer une faiblesse dans le système).

Pour les sommets tests, une notion de criticité peut aussi être introduite.



(a) Extrait du logigramme “Bourrage sequence vidage” effectué avec Atanor.



(b) La représentation en graphe en niveaux.

Les rectangles sur chacun des dessins représentent des parties identiques dupliquées. Dans le logigramme, ils permettent de situer ces parties.

FIG. 6.12 – Illustration de la duplication de sous-arbres du logigramme sur deux extraits de la représentation d'un même processus métier.

Un sommet test est qualifié de critique si une mauvaise réponse entraîne un diagnostic très long à effectuer alors que le même diagnostic aurait pu être effectué immédiatement. De tels sommets sont facilement visibles sur le graphe : le test "Changement du relais RCD" situé au bas du troisième niveau de la figure 6.11 est critique. Dans le meilleur des cas, ce sommet est facilement atteint en deux étapes intermédiaires : "Relais RDD" puis "Alimentation de la bobine du relais RCD". Si l'utilisateur se trompe de réponse, au lieu d'arriver au diagnostic final "Mauvais fonctionnement du relais RCC" instantanément, il sera dirigé vers le sommet échec afin de passer sur la branche numérotée 3 du sommet module. Le même diagnostic sera alors effectué en 6, 7 ou 8 tests selon les réponses données. Il est même possible que l'utilisateur en arrive à un diagnostic final faux avec toutes les conséquences que l'on peut imaginer.

6.5 Conclusion

Dans ce chapitre, nous avons présenté le système de gestion des connaissances Atanor et plus particulièrement le module *Expert* ainsi que la modélisation des connaissances utilisées sous la forme de logigrammes. Nous avons montré à l'aide de plusieurs exemples les limites de ce modèle dont la principale est une redondance parfois importante des sommets ou même de sous-arbres. Ces redondances entraînent une complexité accrue des logigrammes et des difficultés lors de la conception des modèles par les experts.

Le codage des modèles dans le serveur de connaissance étant déjà basé sur des graphes, qui sont connus comme des outils performants de représentation, nous avons proposé un nouveau modèle de représentation visuelle des connaissances qui utilise des graphes en niveaux. Ce modèle, baptisé Graph'Atanor, supprime toutes les redondances de sommets et facilite, selon les premiers retours d'expérience, la lecture des modèles. Pour s'intégrer dans les spécifications du serveur de connaissances, l'implémentation de Graph'Atanor, présentée dans l'annexe B page 149 utilise des technologies actuelles comme les langages XML et SVG.

Chapitre 7

Conclusion

La problématique de cette thèse, réalisée grâce à un contrat *CIFRE* avec l'entreprise KnoweSia SAS, porte sur l'amélioration de la représentation visuelle des connaissances au sein du serveur de connaissances Atanor en se focalisant sur la phase de capitalisation des connaissances des experts. La visualisation des connaissances est un domaine en plein essor dont l'intérêt est fondamental dans un processus de GC. Elle doit être considérée avec attention afin de permettre aux experts d'utiliser convenablement le serveur de connaissances et ainsi éviter les erreurs dues à des représentations visuelles inadaptées.

Dans la première version d'Atanor, les représentations visuelles sont basées sur un modèle de logigramme. Ce modèle s'est avéré difficile à utiliser à cause de nombreuses redondances de sommets. Nous avons donc proposé un nouveau modèle de représentation visuelle des connaissances basé sur des graphes en niveaux : le modèle Graph'Atanor. Et, nous avons proposé un nouvel algorithme, basé sur un AG pour aborder le problème de tracé. Le choix de la méthode nous a conduit à analyser les caractéristiques de l'espace de recherche de notre problème de tracé.

Les apports de la thèse

Les apports de cette thèse peuvent se résumer en quatre grandes parties :

1. le problème de représentation visuelle des connaissances dans Atanor nous a tout d'abord conduit à étudier les différentes familles de représentations visuelles des graphes ;
2. nous avons étudié les caractéristiques des espaces de recherche associés au problème spécifique du tracé des graphes en niveaux avec minimisation du nombre de croisements d'arcs ;

3. ces analyses nous ont conduit à développer un algorithme génétique hybridé efficace pour tracer les graphes. Nous avons proposé une extension pour le problème dynamique et au-delà des applications pratiques, nous avons étudié le comportement de l'AG pour des problèmes de grandes tailles ;
4. nous avons confronté Graph'Atanor avec le modèle des logigrammes actuellement utilisé dans le serveur de connaissances. Nous avons montré les différents apports de notre modèle de graphes par rapport au modèle des logigrammes qui est basé sur des représentations d'arbres.

Etude et classification des représentations visuelles des graphes

Selon le type de la structure à tracer, différentes méthodes sont envisageables. Nous avons présenté quatre grandes familles de tracés basées sur des représentations classiques sommets-arcs :

1. le tracé statique qui est le problème le plus classique ;
2. le tracé dynamique qui est un problème important dans le cadre d'un processus de tracé de graphes interactifs ;
3. le tracé des grands graphes pour représenter des structures ne pouvant pas être tracées sur un support de taille standard ;
4. les représentations en 3D qui deviennent de plus en plus populaires mais dont la valeur ajoutée est encore un sujet d'étude.

Nous avons aussi présenté quelques techniques qui n'utilisent pas les représentations sommets-arcs et les langages de description de graphes les plus utilisés dans les logiciels.

Etude approfondie du problème de tracé des graphes en niveaux

En utilisant les différentes contraintes à respecter pour obtenir des tracés exploitables par un utilisateur, nous obtenons un problème d'optimisation combinatoire qui consiste à trouver un ordre optimal des sommets dans chacun des niveaux du graphe. L'étude bibliographique des différentes méthodes de tracé existantes montre que la structure du paysage de recherche a rarement été étudiée. Cette étude est fondamentale car les méthodes d'optimisations sont efficaces si la méthode utilisée est bien adaptée aux caractéristiques du paysage.

Nous avons utilisé une modélisation originale des paysages de recherche par un graphe en niveaux. Nous avons montré que ces paysages sont fortement multimodaux et que la modélisation en graphe permet de mettre en avant une certaine structure des paysages. En conclusion, il apparaît qu'une méthode de résolution bénéficiant d'une forme d'apprentissage implicite telle que les algorithmes génétiques est tout indiquée.

Un algorithme génétique hybridé pour le tracé des graphes en niveaux

Nous avons développé un algorithme génétique hybridé (AGH) pour résoudre le problème de tracé. Nous avons justifié l'ensemble des valeurs des différents paramètres et le rôle de chacun des opérateurs. Les comparaisons effectuées avec d'autres méthodes déjà publiées montrent que AGH dépasse en terme de qualité des résultats les heuristiques classiques ainsi que la meilleure métaheuristique connue à ce jour basée sur une recherche Tabou. AGH est également meilleur que les descentes à départs multiples pour des graphes d'ordres et de densités compatibles avec un affichage sur un support de taille standard.

Nous avons aussi réalisé deux extensions de cet algorithme. La première porte sur le tracé dynamique d'un graphe sur un intervalle de temps restreint. Les résultats obtenus avec un simulateur automatique de tracé dynamique sont encourageants et montrent que notre approche permet d'obtenir des tracés successifs visuellement proches et qui restent lisibles et compréhensibles par les utilisateurs. La deuxième extension pour des graphes de grandes tailles qui dépassent de loin les tailles rencontrées dans la pratique, montre que les descentes obtiennent alors de meilleurs résultats. Ceci semble dû à un changement de structure du paysage et à la présence de très nombreux optima locaux de qualité comparable.

Le modèle Graph'Atanor

Nous avons proposé Graph'Atanor qui est un nouveau modèle de représentation visuelle des connaissances pour le serveur Atanor. Un de ses avantages est qu'il correspond au modèle formel utilisé pour le stockage des connaissances. Les comparaisons effectuées avec le modèle des logigrammes montrent que l'utilisation de Graph'Atanor est plus intuitive pour les experts en supprimant les redondances de sommets. De plus, il permet une meilleure exploitation des modèles de connaissances en faisant notamment apparaître les caractéristiques importantes des modèles. Graph'Atanor ouvre

des perspectives intéressantes sur l'analyse des traces des utilisateurs et de leur comportement.

Perspectives

En plus de la nécessaire validation par les experts du modèle Graph'Atanor et de ses applications dans d'autres contextes applicatifs, nous pouvons mettre en avant trois axes majeurs d'extensions :

1. l'exploitation du modèle Graph'Atanor a des conséquences notables sur la gestion des ressources humaines associées aux modèles de connaissances ;
2. l'extension de AGH pour le tracé dynamique est à développer afin de produire une plate-forme complète de construction, de modification et d'analyse de modèles de connaissances basés sur des graphes en niveaux ;
3. la deuxième extension de AGH pour les graphes de grandes tailles, nous incite aussi à développer ce type de représentation puisque la taille des modèles utilisés en visualisation de connaissances, et plus généralement en visualisation d'information, ne cesse de croître.

Applications possibles en gestion des ressources humaines

La suppression des redondances de sommets dans Graph'Atanor permet une meilleure exploitation des représentations visuelles par les utilisateurs. Cette meilleure exploitation a des conséquences dans le domaine des ressources humaines pour les différentes classes d'utilisateurs d'Atanor. Dans le cas de l'aide à la maintenance, en faisant apparaître les sommets critiques sur les tracés, en effectuant un comptage des passages dans chaque sommet ou encore pour un même diagnostic, en analysant les différentes traces des utilisateurs, il est possible de :

1. faire ressortir les faiblesses du système (sommets ou parties du graphe très utilisés) qui nécessitent une surveillance particulière ;
2. mieux former les utilisateurs du système en relevant les défauts qui reviennent le plus souvent ou en les sensibilisant sur les sommets critiques ;
3. aider les utilisateurs à comprendre le fonctionnement de la machine et le mode de pensée des experts en visualisant directement les graphes.

L'affichage de ces informations sur les graphes peut se faire en modulant la largeur et la couleur des arcs en fonction du nombre de passages, en superposant différents tracés ou encore en effaçant des parties du tracé. En plus d'être à disposition des managers, ces informations pourraient aussi être utilisées dans un nouveau module *Pédagogique*. Ce module servirait d'outil pour la formation des utilisateurs et permettrait aussi aux experts de tester des mises à jour sur les utilisateurs du système avant de les implémenter.

Prise en compte du côté dynamique

Les résultats obtenus avec l'extension pour le tracé dynamique de AGH, nous incite à développer cette approche. De façon générale, tout processus de construction de graphe dans un SGC est dynamique. Dans le cadre de l'application à Atanor, nous devons encore développer une interface complète pour le module *Expert* qui permette de construire les graphes de façon dynamique. Pour ne pas perturber le processus de construction, il ne faut pas présenter à l'utilisateur un tracé optimisé à chaque étape. Il faut par exemple que l'algorithme d'optimisation tourne en tâche de fond à chaque modification pour conserver en mémoire un tracé optimisé. Quand l'utilisateur ajoute un sommet sur sa représentation, ce sommet est aussi ajouté sur le tracé maintenu par l'algorithme d'optimisation qui peut ainsi être rapidement remis à niveau. Quand l'utilisateur le décide, la solution optimisée lui est présentée très rapidement. Pour limiter l'effort cognitif de l'utilisateur, la transition entre les tracés peut aussi être animée.

Changement d'échelle

Lorsque l'on considère en particulier l'intégralité des bases de connaissances, les représentations du modèle Graph'Atanor peuvent ne pas tenir correctement sur un support de taille standard tout en conservant leur lisiblité. Par exemple, le graphe qui permet de représenter l'ensemble des pannes de la machine de tri de courrier comporte 553 sommets, 625 arcs et 14 niveaux. Il permet d'apporter des informations supplémentaires intéressantes (par exemple, un même diagnostic peut se retrouver dans plusieurs logigrammes différents). Mais même après optimisation du tracé avec AGH, il reste encore plus de 900 croisements d'arcs ; donc la figure est difficilement exploitable. Conformément aux résultats obtenus avec l'extension de AGH pour les grands graphes, il faudrait maintenant développer, en complément à ce que nous venons de faire, une méthode de visualisation adaptée à une structure de taille plus importante.

Bibliographie

[1] G. Aissaoui, D. Genest, et S. Loiseau. « Le modèle des cartes cognitives de graphes conceptuels : un modèle graphique d'aide à la prise de décision ». *Actes 2^e journées francophones Modèles Formels de l'Interaction (MFI)*, pages 243–248. Cepaduès, 2003.

[2] M. Alavi et D.E. Leidner. « Review : Knowledge Management and Knowledge Management Systems : Conceptual foundations and research issues ». *Mis Quarterly*, 25(1) :107–136, 2001.

[3] D. Auber. « Tulip ». *Proc. of the 9th Symp. Graph Drawing*, pages 335–337. LNCS 2265 - Springer Verlag, 2001.

[4] D. Auber. « *Outils de visualisation de larges structures de données* ». Thèse de doctorat, Université dde Bordeaux I, 2002.

[5] G. Aubertin, I. Boughzala, et J.L. Ermine. Cartographie de connaissances critiques. *Revue des sciences et technologies de l'information (RSTI) – Extraction et gestion des connaissances (EGC 2003)*, vol. 17 de *Extraction des connaissances et apprentissage*, pages 495–502. Hermès, 2003.

[6] J.-P. Barthélemy et E. Mullet. « A model of selection by aspects ». *Acta Psychologica*, 79 :1–19, 1992.

[7] O. Bastert et C. Matuszewski. Layered Drawings of Digraphs. M. Kaufmann et D. Wagner, eds., *Drawing Graphs*, vol. 2025 de *LNCS*, pages 87–120. Springer-Verlag, 2001.

[8] V. Batagelj et A. Ferligoj. Clustering relational data. W. Paul, O. Opitz, et M. Schader, eds., *Data Analysis*, pages 3–25. Springer, 2000.

[9] C. Berge. *Graphes et hypergraphes*. Dunod, 1973.

[10] B. Le Blanc, C. Obernesser, et B. Claveri. Validation et évaluation cognitives de techniques de navigation et de visualisation de donnée. *Revue des sciences et technologies de l'information (RSTI) – Extraction et gestion des connaissances (EGC 2003)*, vol. 17 de *Extraction des connaissances et apprentissage*, pages 93–104. Hermès, 2003.

[11] C. Blum et A. Roli. « Metaheuristics in Combinatorial Optimization : Overview and Conceptual Comparison ». *ACM Computing Surveys*, 35(3) :268–308, 2003.

[12] J. Branke. Dynamic Graph Drawing. D. Wagner M. Kaufmann, ed., *Drawing Graphs : methods and models*, pages 228–246. Springer, 2001.

[13] J. Branke, F. Bucher, et H Schmeck. « Using Genetic Algorithms for Drawing Undirected Graphs ». *Proc. of the* 3^{rd} *nordic workshop on Genetic Algorithms and their applications*, pages 193–206. Finnish Artificial Intelligence Society, 1997.

[14] S. Bridgeman et R. Tamassia. « A user study in similarity measures for graph drawing ». *J. of Graph Algorithms and Applications*, 6(3) :225–254, 2002.

[15] R.A. Burkhard. « Learning from Architects : The difference between knowledge visualization and information visualization ». *Proc. of the* 8^{th} *Int. conf. on Information Visualisation (IV)*, pages 519–524, 2004.

[16] T. Buzan et B. Buzan. *The Mind Map Book*. Plume Books, 1996.

[17] K.F. Böhringer et F.N. Paulisch. « Using constraints to achieve stability in automatic graph layout algorithms ». *Proc. of CHI'90*, pages 43–51. ACM, 1990.

[18] S. K. Card, J. D. Mackinlay, et B. Shneiderman. *Readings in Information Visualization ; Using vision to think*. Morgan Kaufmann, 1999.

[19] M.J. Carpano. « Automatic display of hierarchized graphs for computer aided decision analysis ». *IEEE Trans. Syst., Man, Cybern.*, 10(11) :705–715, 1980.

[20] C.A. Coello Coello et G.B. Lamont, eds.. *Applications of Multi-Objective Evolutionary Algorithms*, vol. 1 de *Advances in Natural Computation*. World Scientific, 2004.

[21] C.A. Coello Coello, D.A. Van Veldhuizen, et G.B. Lamont. *Evolutionary Algorithms for Solving Multi-Objective Problems*, vol. 5 de *Genetic Algorithms and Evolutionary Computation*. Kluwer Academic Publishers, 2002. ISBN 0-3064-6762-3.

[22] R.-F. Cohen, G. Di Battista, R. Tamassia, et I.-G. Tollis. « A framework for dynamic graph drawing ». *Proc. of ACM Symp. on Computational Geometry*, pages 261–270. ACM Press, 1992.

[23] « Colloque cartographie de l'information : de la visualisation à la prise de décision dans la veille et le management de la connaissance », 2002. ESIEE, Noisy Le Grand (93).

[24] O. Corby et R. Dieng. « WebCokace : a commonKADS expertise model web server ». *Proc. of the* 11^{th} *Workshop on Knowledge acquisition, Modeling and Management (KAW)*, 1998.

[25] R. Davidson et D. Harel. « Drawing Graphs Nicely using Simulated Annealing ». *ACM Trans. on Graphics*, 15(4) :301–331, 1996.

[26] L. Davis. A genetic algorithms tutorial. L. Davis, ed., *Handbook of Genetic Algorithms*, pages 1–101. Van Nostrand Reinhold, 1991.

[27] G. Di-Battista, P. Eades, R. Tamassia, et I.-G. Tollis. *Graph drawing – Algorithms for the visualization of graphs.* Prentice-Hall, 1999.

[28] R. Dieng-Kuntz, O. Corby, F. GAndon, A. Giboin, J. Golebiowska, N. Matta, et M. Ribière. *Méthodes et outils pour la gestion des connaissances.* Dunod, 2001. 2^{eme} édition.

[29] H. A. D. do Nascimento et P. Eades. « A Focus and Constraint-Based Genetic Algorithm for Interactive Directed Graph Drawing. ». *HIS : Proc. of Hybrid Intelligent Systems*, pages 634–643, 2002.

[30] H.A.D. do Nascimento et P. Eades. « User hints for directed graph drawing ». *Proc. of the* 9^{th} *Graph Drawing Conf (GD'01)*, vol. 2265 de *LNCS*, pages 205–219, 2001.

[31] M. Dodge et R. Kitchin. *Atlas of Cyberspace.* Addison-Wesley, 2000.

[32] P. Eades et D. Kelly. « Heuristics for reducing crossings in 2-layered networks ». *Ars Combinatorics*, 21 :89–98, 1986.

[33] P. Eades, W. Lai, K. Misue, et K. Sugiyama. « Preserving the mental map of a diagram ». *Proc. of Compugraphics*, pages 24–33, 1991.

[34] P. Eades et K. Sugiyama. « How to draw a directed graph ». *Journal of Information Processing*, 4(13) :424–437, 1990.

[35] P. Eades et N. Wormald. « Edge crossings in drawings of bipartite graphs ». *Algorithmica*, 11 :379–403, 1994.

[36] M. Earl. « Knowledge management strategies : toward a taxonomy ». *J. of Management Information Systems*, 18(1) :215–233, 2001.

[37] J. Ellson, E.R. Gansner, E. Koutsofios, S.C. North, et G. Woodhull. Graphviz and Dynagraph – Static and Dynamic Graph Drawing Tools. M. Jünger et P. Mutzel, eds., *Graph Drawing Software*, pages 127–148. Springer, 2004.

[38] T. Eloranta et E. Mäkinen. « TimGA : A Genetic Algorithm for Drawing Undirected Graphs ». *Divulgaciones Matemáticas*, 9(2) :155–171, 2001.

[39] M. Eppler et R. Burkhard. Knowledge Visualization. D.G. Schwartz, ed., *Encyclopedia of Knowledge Management*. Idea Group, 2005.

[40] J.L. Ermine. *Les systèmes de connaissances, 2^{eme} édition*. Hermès, 2000.

[41] T. Eschbach, W. Günther, R. Drechsler, et B. Becker. « Crossing reduction by windows optimization ». *Proc. of the 10^{th} Int. Symposium on Graph Drawing*, vol. 2528 de *LCNS*, pages 285–294. Springer Verlag, 2002.

[42] T.A. Feo et M.G.C. Resende. « Greedy Randomized Search Adaptive Procedures ». *J. of Global Optimization*, 6 :109–133, 1995.

[43] D. Follut, F. Guillet, P. Vandekerckove, et J. Philippe. « Samanta : towards using virtual reality in a computer-assisted environment for the maintenance of postal sorting machines ». 1^{st} *French-British Int. Workshop on Virtual Reality*, 2000.

[44] F. Fürst. « *Contribution à l'ingénierie des ontologies : une méthode et un outil d'opérationalisation* ». Thèse de doctorat, Université de Nantes, 2004.

[45] E. R. Gansner et S. C. North. « An open graph visualization system and its applications to software engineering ». *Software - Practice & Experience*, 30(11) :1203–1233, 2000.

[46] E.R. Gansner, Y. Koren, et S. North. « Topological Fisheye Views for Visualizing Large Graphs ». *IEEE Transactions on Visualization and Computer Graphics*, 11(4) :457–468, 2005.

[47] E.R. Gansner, S.C. North, et K.P. Vo. « A program that draws directed graphs ». *Software - Practice and Experience*, 18(11) :1047–1062, 1988.

[48] M.R. Garey et D.S. Johnson. « Crossing number is NP-complete ». *J. Algebraic Discrete Methods*, 4(3) :312–316, 1983.

[49] M. Ghoniem, J.D. Fekete, et P. Castagliola. « A Comparison of the readability of graphs using node-link and matrix-based representations ». *Proc. of IEEE Symp. on Information Visualization*, pages 17–24, 2004.

[50] F. Glover. « Tabu Search - Part I ». *ORSA J. on Computing*, 1(3) :190–206, 1989.

[51] D.-E. Goldberg. *Genetic algorithms in search, optimization and machine learning*. Addison-Wesley, 1989.

[52] D.E. Goldberg. *The Design of Innovation.* Kluwer Academic Publishers, 2002.

[53] L.J. Groves, Z. Michalewicz, P.V. Elia, et C.Z. Janikow. « Genetic Algorithms for Drawing Directed Graphs ». *Proc. of the 5th Int. Symp. on Methodologies for Intelligent Systems*, pages 268–276. Elsevier, 1990.

[54] F. Guillet, D. Follut, V. Philippé, et J. Philippé. « ATHANOR : Une approche pour la gestion des connaissances de maintenance sur les systèmes complexes ». *1^{ere} journée Systèmes d'information pour l'aide à la décision en ingénierie système (JESIADIS)*, pages 41–54, 2002.

[55] F. Guillet, D. Follut, P. Vandekerckhove, et J. Philippe. « Un serveur de connaissances dans un contexte de maintenance appliquée aux machines de tri postal ». *Journées Internationales Ingénierie de systèmes et NTIC (NimesTIC'2000)*, pages 2–11, 2000.

[56] M.C. Hao, U. Dayal, P. Garg, et V. Machiraju. « Visualization of high-hensity 3D graphs using non-linear visual space transformations ». *Proc. of SPIE conf. on Visualization and Data Analysis*, vol. 4665, pages 269–275, 2002.

[57] W. He et K. Marriot. « Constrained Graph Layout ». *Constraints : An Int. Journal*, 3 :289–314, 1998.

[58] R.J. Hendley, N.S. Drew, A.M. Wood, et R. Beale. « Narcissus : Visualising Information ». *Information Visualization'95*, pages 90–96, 1995.

[59] I. Herman et M.S. Marshall. « GraphXML - An XML-based graph description format ». *Proc. of the Symp. on Graph Drawing*, pages 52–62, 2000.

[60] I. Herman, G. Melançon, et M.S. Marshall. « Graph visualization and navigation in information visualization : a survey ». *IEEE Trans. on Visualization and Computer Graphics*, 6(11) :24–43, 2000.

[61] M.H.W. Hobbs et P.J. Rodgers. « Representing Space : A Hybrid Genetic Algorithm for Aesthetic Graph Layout ». *FEA'98 Frontiers in Evolutionary Algorithms in Proc. of JCIS'98 The 4th Joint Conf. on Information Sciences*, vol. 2, pages 415–418, 1998.

[62] J. Holland. *Adaptation in natural and artificial systems.* Ann Arbor : The University of Michigan Press, 1975.

[63] M.L. Huand, P. Eades, et J. Wang. « Online animated graph drawing using a modified spring algorithm ». *Proc. of Australian Computer Science conf. (ACSC)*, 20(1) :17–28, 1998.

[64] B. Johnson et B. Shneiderman. « Tree-Maps : A space-filling approach to the visualization of hierarchical information structures ». *Proc. of the IEEE* 2^{nd} *conf. on Visualization*, pages 284–291. IEEE computer society press, 1991.

[65] T. Jones. « *Evolutionary Algorithms, Fitness Landscapes and Search* ». Thèse de doctorat, University of New Mexico, Alburquerque, 1995.

[66] T. Jones et S. Forrest. « Fitness distance correlation as a measure of problem difficulty for genetic algorithms ». *Proc. of the Sixth Conf. on Genetic Algorithms*, pages 184–192. Morgan Kaufmann, 1995.

[67] T. Jones et S. Forrest. « Genetic Algorithms and Heuristic Search ». *Santa Fe Institute Tech. Report 95-02-021*. Santa Fe Institute, 1995.

[68] K. De Jong, R. Poli, et J. Rowe, eds.. *Foundations of Genetic Algorithm*, vol. 7. Elsevier, 2003.

[69] M. Jünger et P. Mutzel. « 2-Layer straightline crossing minimization : performance of exact and heuristic algorithms ». *J. of Graph Algorithms and Applications*, 1(1) :1–25, 1997.

[70] H. Kargupta, K. Deb, et D.E. Goldberg. « Ordering genetic algorithms and deception ». *Proc. Parallel Problem Solving from Nature*, vol. 2, pages 47–56. Elsevier Sc., 1992.

[71] M. Kaufmann et D. Wagner, eds.. *Drawing Graphs : Methods and Models*, vol. 2025 de *Lecture Notes in Computer Science*. Springer, 2001.

[72] C. Keskin et V. Vogelmann. « Effective visualization of hierarchical graphs with the cityscape metaphor ». *Proc. of workshop on New paradigms in information visualization and manipulation*, pages 52–57. ACM Press, 1997.

[73] S. Kirkpatrick, C.D. Gelatt, et M.P. Vecchi. « Optimization by Simulated Annealing ». *Science*, 220(4598) :671–680, 1983.

[74] E. Kleiberg, H. Van De Wetering, et J.J.V. Wijk. « Botanical visualization of huge hierarchies ». *Proc. of the IEEE Symp. on Information Visualization (Infovis'01)*, pages 87–94, 2001.

[75] D.E. Knuth. « Computer draw flowcharts ». *Communications of the ACM*, 6(9) :555–563, 1963.

[76] D. Kobler et A. Tettamanzi. « Recombination Operators for Evolutionary Graph Drawing ». *Proc. of Parallel Problem Solving from Nature PPSN-V*, vol. 1498 de *Lect. Notes in Comp. Sc.*, pages 988–997. Springer-Verlag, 1998.

[77] K. Koffka. *The principles of Gestalt Psychology.* Harcourt Brace, 1935.

[78] C. Kosak, J. Marks, et S. Shieber. « A Parallel Genetic Algorithm for Network-Diagram Layout ». *Proc. of the 4th Int. Conf. on Generic Algorithms*, pages 458–465. Morgan-Kaufmann, 1991.

[79] C. Kosak, J. Marks, et S. Shieber. « Automating the Layout of Network Diagrams with Specified Visual Organization ». *IEEE Trans. on System, Man and Cybernetics*, 24(3) :440–454, 1994.

[80] P. Kuntz, R. Lehn, et H. Briand. « Dynamic rule graph drawing by genetic search ». *Proc. of the IEEE Int. Conf. on System Man and Cybernetic*, vol. 4, pages 2481–2486, 2000.

[81] M. Laguna. « A Guide to implementing Tabu Search ». *Investigación Operativa*, 4(1) :5–25, 1994.

[82] M. Laguna et R. Marti. « GRASP and path relinking for 2-layer straight line crossing minimization ». *INFORMS J. on Computing*, 11 :198–204, 1999.

[83] M. Laguna, R. Marti, et V. Valls. « Arc crossing minimization in hierarchical design with Tabu Search ». *Computers and Operations Res.*, 24(12) :1175–1186, 1997.

[84] B. Landgraf. 3D graph drawing. M. Kaufmann et D. Wagner, eds., *Drawing Graphs : Methods and models*, pages 172–192. Springer, 2001.

[85] F. Lehmann. Semantic Network. F. Lehmann, ed., *Semantic Networks in Artificial Intelligence*, vol. 24 de *Int. Series in Modern Applied Mathematics and Computer Science*, pages 1–50. Pergamon Press, 1992.

[86] R. Lehn. « *Un système interactif de visualisation et de fouille de règles pour l'extraction de connaissances dans les bases de données* ». Thèse de doctorat, Université de Nantes, Nantes, 2000.

[87] R. Lehn et P. Kuntz. « A contribution to the study of the fitness landscape for a graph drawing problem ». *Applications of Evolutionary Computing - Proc. of EvoCOP'01*, pages 172–181. LNCS, Springer, 2001.

[88] N. Limnios. *Arbres de défaillances (2^{e} Ed.).* Hermes, 2005.

[89] E. Makinen et M. Sieranta. « Genetic algorithms for drawing bipartite graphs ». *Int. J. of Comput. Math.*, 53(3) :157–166, 1994.

[90] S. Marshall. « *Methods and tools for the visualization and navigation of graphs* ». Thèse de doctorat, Université de Bordeaux I, 2001.

[91] R. MARTÍ. « A Tabu Search algorithm for the bipartite drawing problem ». *European journal of Operational Research*, 106 :558–569, 1998.

[92] R. MARTI. « Arc crossing minimization in graphs with GRASP ». *IIE Trans.*, 33(10) :913–919, 2001.

[93] R. MARTÍ et M. LAGUNA. « Heuristics and Meta-heuristics for 2-layer Straight Line Crossing Minimization ». *Discrete Applied Mathematics*, 127(3) :665–678, 2003.

[94] T. MASUI. « Graphic Object Layout with Interactive Genetic Algorithms ». *Proc. of the 1992 IEEE Workshop on Visual Languages*, pages 74–80. IEEE Comp. Soc. Press, 1992.

[95] C. MATUSZEWSKI, R. SCHÖNFELD, et P. MOLITOR. « Using sifting for k-layer straightline crossing minimization ». *Proc. of the* 7^{th} *Int. Symposium on Graph Drawing*, pages 217–224. LNCS 1731, Springer Verlag, 1999.

[96] K. MEHLORN et S. NÄHER. « LEDA, A platform for combinatorial and geometric computing ». *Communications of the ACM*, 38(1) :96–102, 1995.

[97] M. MINOUX et G. BARTNIK. *Graphes, algorithmes, logiciels*. Dunod Informatique, 1986.

[98] M. MITCHELL, S. FORREST, et J. HOLLAND. « The royal road for genetic algorithms : fitness landscape and GA performances ». *Proc. of the* 1^{st} *Eur.Conf. on Artificial Life*, pages 245–254, 1992.

[99] M. MITCHELL, J. HOLLAND, et S. FORREST. When will a genetic algorithm outperform hill climbing ? J. COWAN, G. TESAURO, et J. ALSPECTOR, eds., *Advances in Neural Information Processing Systems*, vol. 6, pages 51–58. Morgan Kauffman, 1994.

[100] T. MUNZNER. « H3 : Laying out large directed graphs in 3D hyperbolic space ». *Proc. of IEEE Symp. on Information Visualization*, pages 2–10, 1997.

[101] T. MUNZNER. « Drawing large graphs with h3 viewer and site manager ». *Proc. of the* 6^{th} *Symp. Graph Drawing*, pages 384–396. LNCS 1547 - Springer Verlag, 1998.

[102] T. MUNZNER. « *Interactive Visualization of Large Graphs and Networks* ». Thèse de doctorat, Stanford University, 2000.

[103] P. MUTZEL et M. JÜNGER. *Graph drawing software*. Springer Verlag, 2003.

[104] S.C. NORTH et G. WOODHULL. On-line Hierarchical Graph Drawing. *Revised papers from the* 9^{th} *Int. Symp. on Graph Drawing*, vol. 2265 de *LCNS*, pages 232–246. ACM, 2001.

[105] A. Ochoa-Rodrìguez et A. Rosete-Suàrez. « Automatic Graph Drawing by Genetic Search ». *Proc. of the 11th Int. Conf. on CAD, CAM, Robotics and Manufactories of the Future*, pages 982–987, 1995.

[106] A. Ouni et A. Dudezert. « Etat de l'art des approches de définition du Sytème de Gestion des Connaissances ». *9^{e} colloque de l'AIM : Systèmes d'information : perspectives critiques*, 2004.

[107] J. M. Penalva. « Connaissances actionnables et intelligence collective ». *Journées Internationales Ingénerie de systèmes et TIC (Nîmes-TIC'2000*, pages 2–10, 2000.

[108] D.T. Pham, S.S. Dimov, et B.J. Peat. « Intelligent Product Manuals ». *Journal of Engineering Manufacture, Proc. of the Institution of Mechanical Engineers*, 214(B5) :411–419, 2000. Partie B.

[109] J.Y. Prax. *Le Manuel du Knowledge Management.* Dunod, 2003.

[110] H. Purchase. « Which aesthetic has the greatest effect on human understanding ? ». *Proc. Graph Drawing'97*, vol. 1353 de *Lect. Notes in Comp. Sc.*, pages 248–261. Springer Verlag, 1997.

[111] H. Purchase. « Effective information visualisation : a study of graph drawing aesthetics and algorithms ». *Interacting with computers*, 13(2) :147–162, 2000.

[112] A. Reka, H. Jeong, et A.-L. Barabasi. « Diameter of the World-Wide Web ». *Nature*, 401 :130–131, 1999.

[113] J.-M. Renders. *Algorithmes génétiques et réseaux de neurones.* Hermès, 1995.

[114] M.G.C. Resende et C.C. Ribeiro. Greedy Randomized Adaptative Search Procedures. F. Glover et G. Kochenberger, eds., *Handbook of Metaheuristics.* Kluwer, 2003.

[115] K. Risden, C. Czerwinski, T. Munzner, et D. Cook. « An initial examination of ease of use for 2D and 3D information visualization of Web content ». *Int. Journal of Human Computer Studies*, 53(5) :665–714, 2000.

[116] G.G. Robertson, JD. Mackinlay, , et S.K. Card. « Cone trees : Animated 3D visualizations of hierarchical information ». *SIGCHI, Conf. on Human Factors in Computing Systems*, pages 189–194. ACM, 1991.

[117] C. Roche, C. Foveau, et S. Reguigui. La démarche ontologique pour la gestion des compétences et des connaissances. *Revue des sciences et technologies de l'information (RSTI) – Extraction et gestion des*

connaissances (EGC 2005), vol. 2 de *Extraction des connaissances et apprentissage*, pages 359–364. Hermès, 2005.

[118] P. Rosenstiehl. « Claude Berge, ses graphes et hypergraphes ». *Mathématiques et Sciences Humaines*, 160 :7–12, 2002.

[119] A. Rosete et A. Ochoa. « Genetic Graph Drawing ». *Proc. of the* 13^{th} *Int. Conf. of Applications of Artificial Intelligence in Engineering*, pages 37–41, 1998.

[120] A. Rosete-Suárez, A. Ochoa-Rodríguez, et M. Sebag. « Automatic Graph Drawing and Stochastic Hill Climbing ». *Proc. of the Genetic and Evolutionary Conf., GECCO'99*, vol. 2, pages 1699–1706. Morgan Kaufmann, 1999.

[121] A. Rosete-Suárez, M. Sebag, et A. Ochoa-Rodríguez. « A study of Evolutionary Graph Drawing ». Rapport de Recherche 1228, LRI, UMR 8623, Bat. 490, Université Paris-Sud XI, 91405 Orsay-CEDEX, France, Septembre 1999.

[122] L.A. Rowe, M. Davis, E. Messinger, C. Meyer, C. Spirakis, et A. Tuan. « A browser for directed graphs ». *Software - Practice and Experience*, 17(1) :61–76, 1987.

[123] M. Sarkar et M.H. Brown. « Graphical fisheye views ». *Communications of the ACM*, 37(12) :73–84, 1994.

[124] G. Schreiber, B. Wielinga, R. de Hoog, H. Akkermans, et W. Van de Velde. « CommonKADS : a comprehensive methodology for KBS development ». *IEEE Expert*, 9(6) :28–37, 1994.

[125] B. Shneiderman. « The eyes have it : A task by data type taxonomy in information visualization ». *IEEE Symp. on Visual Languages*, pages 336–342. IEEE Press, 1996.

[126] J.F. Sowa. Conceptuals graphs summary. P. Eklund, T. Nagle, J. Nagle, et L. Gerholz, eds., *Conceptual Structures : Current Research and Practice*, pages 3–52. Ellis Horwood, 1992.

[127] T.C. Sprenger, M.H. Gross, D. Bielser, et T. Strasser. « IVORY - An object-oriented framework for physics-based Information Visualization in Java ». *Proc. of IEEE information Visualization'98*, pages 79–86, 1998.

[128] K. Sugiyama, S. Tagawa, et M. Toda. « Methods for visual understanding of hierarchical systems ». *IEEE Trans. Syst., Man, Cybern.*, 11(2) :109–125, 1981.

[129] E.D. Taillard. Optimisation approchée en recherche opérationnelle. J. Teghem et M. Pichot, eds., *Principes d'implémentation des métaheuristiques*, Chapitre 2, pages 57–79. Hermès, 2002.

[130] A. G.B. Tettamanzi. « Drawing graphs with evolutionay algorithms ». *Proc. of Conf. on Adaptative Computing in Design and Manufacture, ACDM*, pages 325–338, 1998.

[131] B.C. Tjaden et G. Wasson. « The Oracle of Bacon », 2000.

[132] J. Utech, J. Branke, H. Schmeck, et P. Eades. « An evolutionary algorithm for drawing directed graphs ». *Proc. of the Int. Conf. on Imaging Science, Systems and Technology*, pages 154–160. CSREA Press, 1998.

[133] V.K. Vassilev, J.F. Miller, et T.C. Fogarty. « Information characteristics and the structure of landscapes ». *Evolutionary Computation*, 8(1) :31–60, 2000.

[134] N. Vergnaud, M. Harzallah, et H. Briand. « Modèle de gestion intégrée des compétences et connaissances ». *Actes Conf. Extraction et Gestion des Connaissances (EGC)*, pages 159–170. Cépaduès Editions, 2004.

[135] C. Ware. *Information visualization : Perception for design*. Interactive Technologies. Morgan Kaufmann, 2000.

[136] C. Ware et G. Franck. « Evaluating stereo and motion cues for visualizing information nets in three dimensions ». *ACM Transactions on Graphics*, 15(2) :121–140, 1996.

[137] C. Ware, H. Purchase, L. Colpoys, et M. McGill. Cognitive Measurements of Graph Aesthetics. *Information Visualization*, vol. 1, pages 103–110. 2002.

[138] J. Warfield. « Crossing theory and hierarchy mapping ». *IEEE J. on Systems, Man and Cybernetics*, 7(7) :505–523, 1977.

[139] D. Whitley et N. Yoo. Modeling simple genetic algorithms for permutation problems. *Fundations of Genetic Algorithms III*. Morgan Kaufmann, 1995.

[140] G.J. Wills. « NicheWorks : Interactive visualization of very large graphs ». *J. of Computational and Graphical Statistics*, 8(2) :190–212, 1999.

[141] A. Winter. Exchanging Graphs with GXL. P. Mutzel, M. Jünger, et S. Leipert, eds., *Graph Drawing : 9^{th} Int. Symp. (GD'01)*, vol. 2265 de *LCNS*, pages 485–500. Springer, 2002.

[142] S. Wright. « The roles of mutation, inbreeding, crossbreeding and selection in evolution ». *Proc. of the 6^{th} Int. Congress of Genetics*, vol. 1, pages 356–366, 1932.

[143] D.A. Zighed et R. Rakotomalala. *Graphes d'induction - Apprentissage et data mining.* Hermès, 2000.

Publications sur les travaux de cette thèse

Chapitre de livre

⋆ P. Kuntz, B. Pinaud et R. Lehn. Elements for the description of fitness landscapes associated with local operators for layered drawings of directed graphs. M.G.C. Resende et J. P. de Sousa, eds., *Metaheuristics : Computer Decision-Making,* vol. 86 de *Applied Optimization*, pages 405–420. Kluwer Academic Publishers, 2004.

Revue internationale

⋆ P. Kuntz, B. Pinaud et R. Lehn. Minimizing crossings in hierarchical digraphs with a hybridized genetic algorithm. *J. of heuristics*, 12(1–2), pages 23–36, 2006, Springer Science+Media B.V.

Revues nationales

⋆ B. Pinaud, P. Kuntz, F. Guillet et V. Philippé. Visualisation en gestion des connaissances, Développement d'un nouveau modèle graphique Graph'Atanor. *Numéro spécial Extraction et Gestion des Connaissances (EGC'2006), Revue des Nouvelles Technologies de l'Information (RNTI)*, pages 311–322, Cépaduès, 2006.

⋆ P. Kuntz, R. Lehn, F. Guillet et B. Pinaud. Découverte interactive de règles d'association via une interface visuelle. P. Kuntz et F. Poulet, eds., *Numéro spécial Visualisation en Extraction des Connaissances, Revue des Nouvelles Technologies de l'Information (RNTI),* Cépaduès, 2006, pages 113–125, sous presse.

Conférences internationales avec actes

⋆ B. Pinaud, P. Kuntz et R. Lehn. Dynamic graph drawing with a hybridized genetic algorithm. I.C. Parmee, ed., *Automatic Computing in Design and Manufacture VI*, pages 365–375, Springer, 2004.

⋆ B. Pinaud, P. Kuntz et R. Lehn. Comparisons of metaheuristics for the layered digraph drawing problem. *Proc of WSEAS Conf. on Simulation, Systems Theory and Systems Engineering*, pages 256–260. WSEAS, 2002.

⋆ R. Lehn, P. Kuntz et B. Pinaud. A descriptive analysis of a landscape for a directed graph layout problem. *Proc. of the* 4^{th} *Metaheuristics Int. Conf.*, vol. 1, pages 18–24, 2001

Conférences nationales avec actes

⋆ B. Pinaud et P. Kuntz. Un guide sur la toile pour sélectionner un logiciel de tracé de graphes. *Actes conférence Veille Stratégique Scientifique et Technologique (VSST)*, vol. 1, pages 339–347, 2004

⋆ B. Pinaud, P. Kuntz, M. Delépine et J. Barberet. GVSR : un annuaire de logiciels de manipulation et d'édition de graphes. *Numéro spécial Extraction et Gestion des Connaissances (EGC'2004)*, *Revue des Nouvelles Technologies de l'Information (RNTI)*, page 416, Poster, Cépaduès, 2004.

Séminaire

⋆ B. Pinaud. Un algorithme génétique hybridé pour le tracé de graphes hiérarchiques. Présentation lors de la dixième Journée Evolutionnaire Trimestrielle (JET-10), juillet 2003.

ANNEXES

Annexe A

L'annuaire GVSR

Le chapitre 2 de la thèse présente des méthodes de tracés de graphes et quelques logiciels qui les implémentent. Sur la Toile, il existe de nombreux autres logiciels. Ils peuvent être soit spécialisés dans un domaine d'application précis ou pour une méthode de tracé précise, soit généralistes en gérant différentes méthodes de tracé sans application préalablement définie sur un domaine particulier. Pour un utilisateur novice dans le domaine du tracé de graphes, il n'est pas nécessairement aisé de retrouver un logiciel bien adapté à sa propre problématique. L'ouvrage de Mutzel et Jünger [103] ainsi que l'annexe A de l'ouvrage de Kaufmann et Wagner [71] peuvent certainement aider à guider l'utilisateur puisqu'ils répertorient des logiciels actuels de visualisation parmi les plus performants. Cependant, ces logiciels puissants n'offrent évidemment pas tous une prise en main immédiate, et les fonctionnalités proposées peuvent dépasser les besoins spécifiques de l'utilisateur ou ne pas répondre aux contraintes d'applications pointues. Différents sites de la Toile présentent également des logiciels généraux ou spécialisés[1]. Mais, il s'agit souvent de listes de pointeurs peu organisées qui nécessitent une investigation importante pour l'analyse de leur contenu. Le site *"Visual Complexity"* propose des aperçus de représentations visuelles existantes[2] en les décrivant sous formes de fiches mais les logiciels utilisés ne sont pas présentés.

Ces limitations nous ont conduit à développer un site Web appelé GVSR[3], disponible à `http://hulk.knowesia.fr`, qui répertorie sous une forme standard les logiciels accessibles sur la Toile. Ce site, qui naturellement se veut évolutif, présente actuellement une cinquantaine de logiciels très divers ré-

[1] `http://rw4.cs.uni-sb.de/users/sander/html/gstools.html` ; `http://www.dia.uniroma3.it/research/ACG.html` ; `http://directory.google.com/Top/Science/Math/Combinatorics/Software/Graph_Drawing/`

[2] `http://www.visualcomplexity.com/vc/index.cfm`

[3] *Graph Visualization Software References*

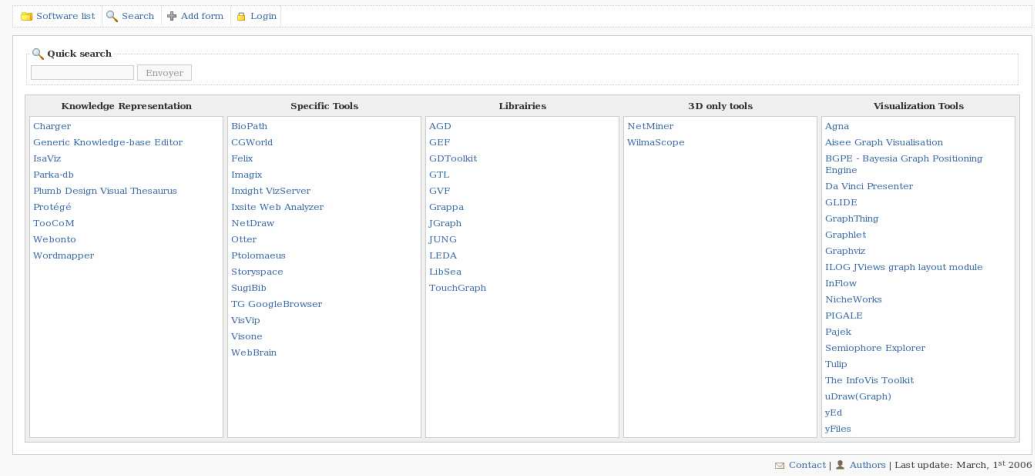


FIG. A.1 – La page d'accueil du site.

partis dans différentes catégories. L'objectif est double : *(i)* permettre, lors d'une étape de découverte, à un utilisateur novice de consulter aisément des exemples de rendus visuels pour affiner ensuite l'expression de ses propres besoins, *(ii)* fournir avec une présentation uniforme des informations précises sur chacun des outils pour le guider dans son choix.

A.1 Description du contenu

La page d'accueil du site regroupe les logiciels dans cinq catégories (figure A.1) : les logiciels spécialisés dans la représentation des connaissances (*"Knowledge Representation"*), les éditeurs de graphes spécialisés dans un domaine particulier qui permettent de construire soi-même interactivement des graphes *("Specific Tools"*), les librairies permettant de se développer facilement et rapidement son propre logiciel en utilisant les implémentations d'algorithmes de tracé présentent dans la librairie (*"Librairies"*), les solutions dédiées aux représentations en 3D (*"3D only tools"*), et les logiciels de tracés généralistes (*"Visualization Tools"*). Un menu est toujours disponible sous le nom du site. Il permet par exemple en cliquant sur *"Add form"* d'accéder à un formulaire permettant de proposer un nouveau logiciel. Avant d'être rendue publique la proposition devra être validée par l'administrateur.

Pour l'utilisateur, la spécificité de l'annuaire GVSR repose sur une lecture homogène de la description de chaque logiciel (voir l'exemple d'une fiche sur la figure A.2). Ainsi, chaque fiche-logiciel se décompose en huit parties (neuf

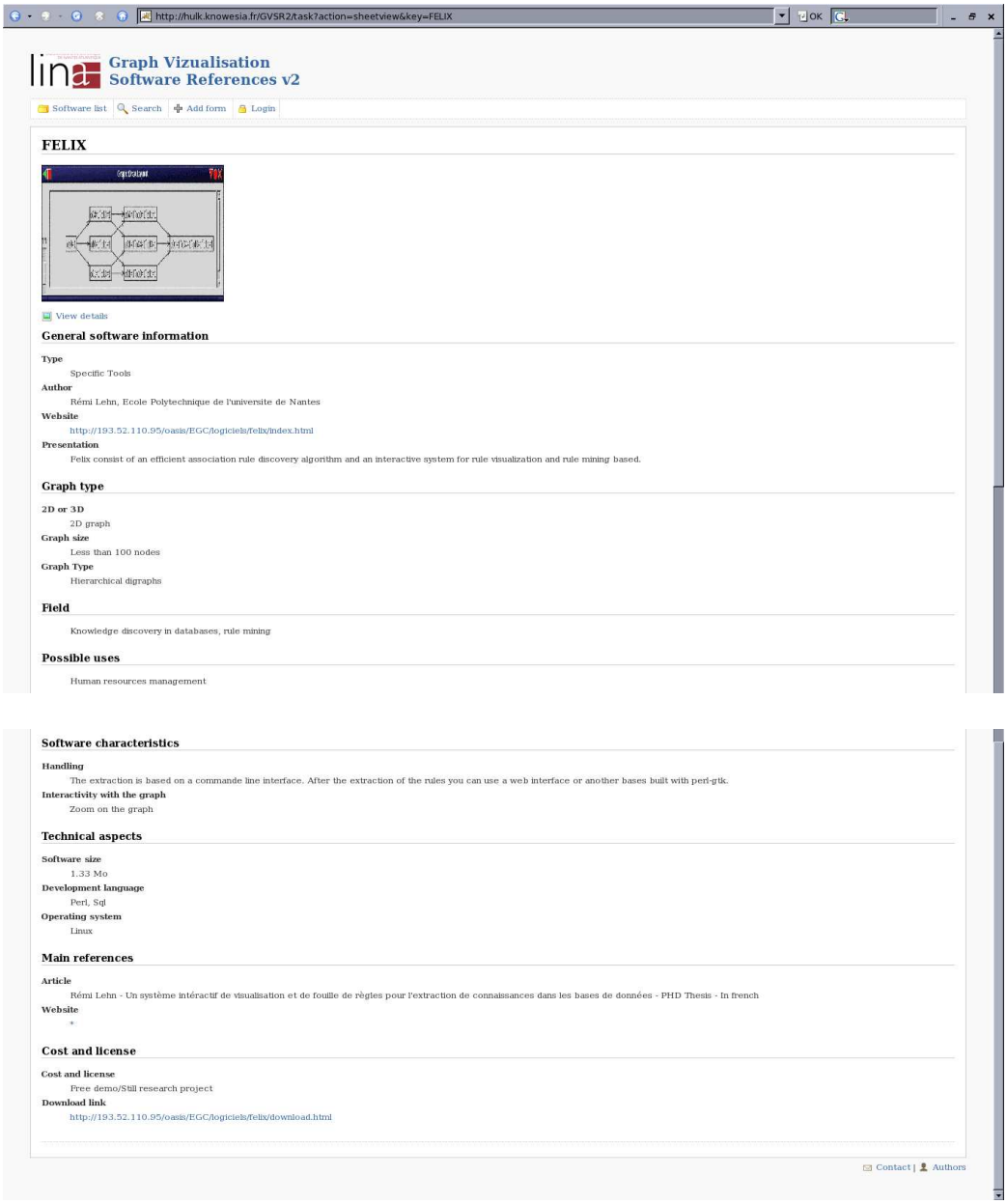


FIG. A.2 – Capture d'écran d'une fiche-logiciel.

pour les librairies) auxquelles s'ajoute une capture d'écran permettant de visualiser sur un exemple les caractéristiques majeures du tracé proposé. Les différentes parties sont :

- *General software information :* une description des informations générales sur le logiciel, à savoir sa catégorie, les différents auteurs (entreprises ou personnes), un lien vers le site web associé et une brève présentation générale.
- *Graph type :* le (ou les) type(s) de graphe(s) manipulé(s) : ordre maximal, type(s), 2D et/ou 3D.
- *Field :* quelques champs d'applications possibles.
- *Possible uses :* quelques exemples connus d'applications.
- *Software characteristics :* les caractéristiques détaillées du logiciel avec son principe succinct d'utilisation et ses fonctions interactives de manipulation des tracés.
- *Technical aspects :* les aspects techniques tels que la taille du logiciel une fois installé, les langages de développement utilisés, l'architecture matérielle et logicielle nécessaire.
- *Main references :* une liste de pointeurs vers les principales références connues (articles et sites web).
- *Main applications :* réservé uniquement pour les librairies. Ensemble de liens vers des applications basées sur cette librairie.
- *Cost and license :* le coût éventuel et la licence d'utilisation.

A.2 Description technique

Les fiches-logiciels sont représentées en utilisant le langage XML qui grâce à son format semi-structuré, nous permet de pouvoir facilement faire évoluer la structure d'une fiche. Pour le stockage, nous utilisons le système de gestion de bases de données (SGBD) XML *"eXist*[4]". Ce SGBD, développé entièrement en JAVA, permet de stocker et d'indexer directement des fichiers XML qui peuvent contenir du texte et éventuellement des données binaires (c'est la cas pour le stockage de l'image associée à chaque fiche). Ils sont regroupés dans des collections qui définissent une arborescence à l'image des répertoires d'un disque dur. Comme dans tous les SGBD, il existe un langage d'interrogation –XQUERY du W3C[5]–, et un langage de mise à jour des données –XUPDATE[6]–, qui agissent sur l'ensemble des documents d'une collection. L'utilisation commune de ces deux langages permet d'avoir des fonctionna-

[4] `http://exist.sourceforge.net`

[5] `http://www.w3.org/TR/xquery`

[6] `http://xmldb-org.sourceforge.net/xupdate`

lités de requêtes et de mise à jour des fiches très proches des possibilités du langage SQL des bases de données relationnelles.

L'interface avec le navigateur de l'utilisateur est réalisée par le logiciel *Tomcat* développé par la "*Apache Software Foundation*"[7]. Tomcat est l'implémentation de référence des technologies *"Java Servlet*[8]" et *"JavaServer Pages*" (JSP)[9] développées par *Sun Microsystems*. Ces technologies permettent de créer facilement des applications basées sur des sites web comportant un contenu dynamique. Grâce à l'utilisation du langage *Java,* les applications développées sont indépendantes des architectures utilisées sur le serveur et les postes clients. GVSR est basé sur la technologie des JSP qui permet d'encapsuler des appels à des classes Java directement dans une page HTML. Ainsi un site utilisant la technologie des JSP comporte : des fichiers JSP qui seront transformés en HTML afin d'être envoyés sur le navigateur de l'utilisateur et un ensemble de classes JAVA qui gère toute la partie dynamique (notamment les différents appels au SGBD).

[7] `http://jakarta.apache.org/tomcat`
[8] `http://java.sun.com/products/servlets`
[9] `http://java.sun.com/products/jsp`

Annexe B

Implémentation de AGH

Le cahier des charges du serveur de connaissances Atanor indique qu'il peut être utilisé au travers du réseau Internet et ne doit pas impliquer de trop gros changements au sein du système d'information servant de support. Il faut, de plus, éviter la lourdeur des "applets" Java actuellement utilisées tout en conservant des possibilités avancées d'interactions avec l'utilisateur pour que les experts puissent continuer à construire les modèles de connaissances directement depuis leur navigateur. Le langage SVG (*Scalable Vector Graphics*)[1], qui est le résultat d'un groupe de travail du W3C[2], respecte parfaitement ces points et a donc été utilisé pour les représentations visuelles de Graph'Atanor. Ce langage permet de faire du dessin vectoriel sur Internet, ce qui a pour avantage :

- l'indépendance par rapport à la résolution utilisée sur le poste client,
- la conservation des données (contrairement au format d'image jpeg),
- d'avoir un principe de fonctionnement équivalent au langage postscript pour n'en citer qu'un.

Un document SVG est interprétable sur de multiples plate-formes et transite sans difficulté sur un réseau car il est basé sur XML.

Langage de description des graphes

Les graphes utilisés par AGH sont simplement décrits par des fichiers XML. Pour un graphe à tracer, ce langage doit permettre simplement de décrire la liste des sommets et des arcs. Des propriétés graphiques, toutes optionnelles, doivent aussi pouvoir être précisées. Toute la gestion des fi-

[1] Le site `http://www.svgopen.org` regroupe les actes des conférences sur SVG ayant eu lieu depuis 2002. Une présentation de ce langage est disponible à l'adresse `http://www.w3.org/2002/Talks/SVG-HongKong-IH/`.

[2] `http://www.w3.org/TR/SVG/`

chiers XML (lecture des fichiers sources, génération des fichiers SVG) a été effectuée en utilisant la librairie libxml2[3]. C'est une librairie pour le langage C très complète permettant de lire, écrire et manipuler des documents et des données en XML.

Le format des fichiers de données, décrit sous la forme d'une grammaire, est présenté ci-dessous :

```
<? xml version="1.0" encoding="encoding" ?>
<graph name="ID">  <!--Nom du graphe-->
<vertex name="ID'  <!--Identifiant du sommet-->
    color="svg_color"  <!--Couleur de remplissage-->
    label="ID"  <!--Nom du sommet à afficher-->
    type="test | diagnostic | module | echec"  <!--type de sommet-->
    fontsize="[0-9]*"  <!--Taille de la police-->
    font="fontname"  <!--Police SVG à utiliser-->
    stroke="svg_color"  <!--Couleur du contour-->
    rank="[0-9]*"/>  <--Niveau du sommet-->
<arc name="ID"  <!--Identifiant de l'arc-->
    from="vertex_name"  <!--Id. du sommet d'origine-->
    to="vertex_name"  <!--Id. du sommet terminal-->
    label="ID"  <!--pour les sommets module : numéro à afficher-->
    color="svg_color"/>  <!--couleur de l'arc-->
</graph>
```

Les différentes clauses *ID* peuvent être remplacées par n'importe quelle chaîne de caractères. Tous les champs sont optionnels sauf ceux notés en gras. Les couleurs peuvent être soit désignées directement en anglais ou alors spécifiées par leur code RVB que l'on écrira : rgb(R,V,B) où R, V et B sont les codes respectifs pour les composantes rouge, verte et bleue. La clause *name* sert à préciser un identifiant utilisé en interne par le programme. Pour les sommets, si la clause *label* n'est pas précisée, l'identifiant est utilisé à sa place. Pour les arcs, *label* n'est utilisé que pour les sommets modules pour afficher le numéro d'ordre. La clause *type* est un raccourci pour donner une esthétique différente selon le type de sommet du modèle expertise à dessiner. L'algorithme de placement des sommets dans les niveaux (ainsi que la suppression des cycles) peut être désactivé en précisant impérativement pour tous les sommets du graphe leur niveau respectif par la clause *rank*.

[3] `http://xmlsoft.org`

Pour avoir une gestion correcte des caractères spéciaux (par exemple les caractères accentués en français), il faut utiliser un encodage des caractères (la clause *encoding*) compatible entre différentes langues et facilement portable. Pour le français, il est par exemple possible d'utiliser ISO–8859–1 ou ISO–8859–15. Si rien n'est précisé, c'est l'encodage par défaut de la libxml2 qui est utilisé, à savoir UTF–8.

Résumé :
Le bon déroulement d'un processus de gestion des connaissances passe par l'utilisation de méthodes efficaces de visualisation qui permettent une compréhension aisée des différents modèles de connaissances utilisés. Les retours d'expériences avec le système de gestion des connaissances Atanor, qui est orienté vers le déploiement des connaissances dans un contexte opérationnel portant sur des systèmes complexes, ont montré que le modèle d'arbres actuellement utilisé pour la visualisation des modèles de connaissances n'est pas intuitif. Des redondances de sommets trop nombreuses peuvent entraîner des difficultés de lecture et cacher des caractéristiques importantes. Pour résoudre ces problèmes nous proposons le modèle Graph'Atanor qui est basé sur des graphes en niveaux.
Le passage au modèle de graphes pose le problème de sa représentation visuelle. Les tracés doivent rester lisibles et compréhensibles par les utilisateurs. Ceci se traduit notamment par le respect de critères esthétiques qui permettent de modéliser un problème d'optimisation combinatoire consistant à trouver un ordre optimal des sommets dans chaque niveau. Pour résoudre ce problème, nous avons développé un algorithme génétique qui possède deux particularités : deux opérateurs de croisements spécifiques et une hybridation par une recherche locale.
Les expérimentations montrent que pour des graphes de taille standard, l'algorithme génétique donne de meilleurs résultats que les autres méthodes que nous connaissons. La comparaison des modèles de représentation des connaissances sur un exemple industriel montre qu'en plus de faciliter la lecture, Graph'Atanor permet de facilement suivre la trace des utilisateurs et de mettre en avant les sommets critiques.
Mots-clés :
Visualisation de connaissances, système de gestion des connaissances, optimisation combinatoire, métaheuristiques, algorithme génétique hybridé, tracés de graphes en niveaux

Abstract :
The smooth development of a knowledge management process is based on the use of efficient visualization methods that enable the user to easily understand the knowledge models which are used. Experience feedback of the knowledge management system Atanor, which is knowledge-deployment-oriented in an operational context for complex systems, showed that the tree model currently used for the visualization of the knowledge models is not intuitive. Too many vertices redundancies can lead to reading difficulties and hide some important characteristics. To solve these problems, we propose the Graph'Atanor model which is based on hierarchical graphs.
Moving to a graph-based model raises the issue of its visual representation. The drawings have to remain readable and understandable by all users. They have to satisfy various criteria such as aesthetic ones which model a combinatorial optimization problem consisting, for each layer, in finding an optimal order of the vertices. To solve this problem, we develop a new hybridized genetic algorithm which follows the basic scheme of a genetic algorithm with two major differences: the use of two problem-based crossovers and a hybridization resulting from a local-search strategy.
Computational experiments for standard size graphs show that the genetic algorithm gives better results than the other methods we know. The comparison of the two knowledge models on an industrial example shows that Graph'Atanor based models are not only easier to read but also allow to follow the trace of the users and show critical vertices.
Keywords :
Knowledge visualization, knowledge management system, combinatorial optimization, metaheuristics, hierarchical graph drawing, hybridized genetic algorithm