Planification et ordonnancement probabilistes sous contraintes temporelles

Bassam Baki

UNIVERSITÉ de CAEN/BASSE-NORMANDIE

U.F.R. : Sciences

ÉCOLE DOCTORALE : SIMEM

THÈSE

présentée par

Bassam Baki

et soutenue

le 30 Novembre 2006

en vue de l'obtention du

DOCTORAT de l'UNIVERSITÉ de CAEN

spécialité : Informatique

(Arrêté du 07 août 2006)

Planification et ordonnancement probabilistes sous contraintes temporelles

MEMBRES du JURY

Abdel-Illah Mouaddib	**Professeur**	**Université de Caen Basse-Normandie**	**(directeur)**
François Charpillet	**Directeur de recherche**	**INRIA Nancy**	**(rapporteur)**
Sylviane R. Schwer	**Professeure**	**Université de Paris Nord Paris**	**(rapportrice)**
François Bourdon	**Professeur**	**Université de Caen Basse-Normandie**	**(examinateur)**
Maroua Bouzid	**Maître de conférences**	**Université de Caen Basse-Normandie**	**(examinatrice)**
Antoni Ligeza	**Professeur**	**Université de Cracow Pologne**	**(examinateur)**
Thierry Vidal	**Maître de conférences**	**ENIT Tarbes**	**(examinateur)**

Mis en page avec la classe thloria.

Remerciements

Je veux remercier tous ceux qui m'ont donné l'occasion de faire ma thèse dans de bonnes conditions, tous ceux qui m'ont aidé, ceux qui m'ont permis d'apprendre, d'approfondir, de progresser, ...

Je tiens à remercier très sincèrement Abdel-Illah Mouaddib, mon directeur de thèse, Professeur à l'université de Caen, pour la confiance qu'il m'a accordée et pour les précieux conseils et l'aide qu'il a bien voulu me prodiguer.

Je remercie également Maroua Bouzid, Maître de conférences à l'université de Caen, qui a co-encadré ma thèse, pour son aide, et ses remarques précieuses tant dans mon travail que dans la rédaction des articles et de la thèse. Je la remercie aussi pour sa gentillesse, sa disponibilité et ses encouragements.

Je remercie Sylviane R. Schwer, Professeure à l'université de Paris Nord, et François Charpillet, Directeur de recherche à l'INRIA, pour avoir accepté de participer à mon jury de thèse et pour leurs remarques qui m'ont été profitables.

J'exprime ma reconnaissance à François Bourdon, Professeur à l'université de Caen, qui me fait l'honneur de présider le jury de ma thèse.

J'adresse mes sincères remerciements à Antoni Ligęza, Professeur à l'université de Cracovie, pour l'intérêt qu'il a porté à mon travail, pour son aide pendant ses séjours au GREYC et pour avoir accepté de participer à mon jury de thèse.

Je remercie également Thierry Vidal, Maître de conférences à l'Ecole Nationale d'Ingénieurs de Tarbes, pour ses conseils précieux lors de notre discussion à Time'04, pour ses remarques qui m'ont été utiles et pour avoir accepté de participer à mon jury de thèse.

J'exprime ma gratitude aux thésards avec qui j'ai partagé le bureau pendant ma thèse, Tuan Dang Nguyen, Jin Yao et Nadia Zerida. Leur gentillesse, leur humour et leurs conseils m'ont été d'un grand apport.

Je voudrais également remercier tous ceux avec qui je partage les repas au RU et les pauses-café, et avec qui j'apprécie les discussions enrichissantes tant du point de vue scientifique que sur le plan humain, tout particulièrement Salah El Falou et Hossam Hanna.

Je remercie également tout le personnel du laboratoire GREYC de m'avoir accueilli et aidé. Surtout les membres de l'équipe MAD, les secrétaires et les administrateurs-système.

Je remercie vivement Pierre Ageron pour son aide précieuse, sa disponibilité et son encouragement tout au long de mon séjours à Caen. Je le remercie pour le temps qu'il m'a accordé pour lire et relire mes articles et ma thèse. Je n'oublierai jamais toutes ses remarques souvent pertinentes.

Un grand merci à Gilles et Jean-Louis pour leur soutien et pour le temps qu'ils ont consacré à la correction de mes articles en anglais.

Merci à mes colocataires Nabil Tahhan et Nicolas Leroux pour leur encouragement et leur enthousiasme quand j'avais eu besoin et aussi pour les moments sympa qu'on a passés ensemble.

Si je commence à écrire les noms de tous ceux qui me rappellent qu'il y a une vie en dehors du GREYC, je ne finirai jamais. Alors, à tous, merci.

Je ne remercierai jamais assez ma famille. Un grand merci à :
- mon père pour son aide, sa confiance, son encouragement et son soutien ; je lui dédie cette thèse ;
- ma belle mère pour les nouvelles du village qu'elle me racontait à chaque appel téléphonique chez moi ;
- mes frères et sœurs Issam, Wissam, Wiam, Ahlam, Nada et Ali pour leur encouragement et la joie que je trouve avec eux pendant le mois de vacances que nous passons chaque été tous ensemble ;
- ma tante Zeinab pour sa confiance.

Enfin, MERCI Fatmé, pour tout.

Résumé

Titre : Planification et ordonnancement probabilistes sous contraintes temporelles

Cette thèse est consacrée au problème de la planification et de l'ordonnancement des tâches sous contraintes temporelles et incertitude. Les contraintes temporelles que nous traitons sont de deux types : qualitatives et quantitatives. L'incertitude sur la durée des tâches se traduit par une distribution de probabilités sur un ensemble fini. Les tâches et les contraintes sont représentées à l'aide d'un graphe ET/OU et les durées des tâches sont pondérées par des probabilités d'exécution. Celles-ci expriment une incertitude sur la connaissance exacte des durées d'exécution des tâches qui ne seront réellement connues que lors de l'exécution effective. Ainsi, une tâche s'exécute durant l'une de ses durées d'exécution possibles avec la probabilité associée à celle-ci. Étant donné ce graphe, notre objectif est de déterminer un plan de tâches qui satisfait toutes les contraintes et qui répond aux critères de choix exigés par l'utilisateur en terme de temps, de coût et de probabilité. L'application de ce plan doit garantir le monde de façon que le but soit atteint tout en satisfaisant les contraintes du domaine. Nous avons appliqué notre méthode de planification à un cas pratique relativement complexe qui concerne la planification d'un ensemble d'agents travaillant ensemble dans un lieu afin d'atteindre un but donné tout en respectant les délais et les contraintes du domaine (temps, coût, probabilité, disponibilité, spécialité, etc.).

Mots-clés : Planification, Ordonnancement, IA, Contraintes temporelles, Incertitude et Agents.

Abstract

Title : Probabilistic Planning and Scheduling under Temporal Constraints

This thesis is devoted to the problem of the planning and scheduling of the tasks under temporal constraints and uncertainty. We deal with two categories of temporal constraints : qualitative and quantitative. Uncertainty over the duration of the tasks is presented in the form of a probability distribution on a finite set. The tasks and the constraints are represented by an AND/OR graph and the durations of the tasks depend on probabilities of execution. Those express an uncertainty on the exact knowledge of the execution time of the tasks which will be really known only during the effective execution. Thus, the execution task takes one of its possible execution durations with a certain probability. Given this graph, our objective is to determine a plan of tasks which satisfies all the constraints and which satisfies the selection criteria required by the user in terms of time, cost and probability. The application of this plan must guaranties that the goal is reached while satisfying the constraints of the environment. We applied our method of planning on an example inspired from a relatively complex real world application which relates to the planning of a set of agents working together in order to achieve a goal while the deadlines and the constraints of the environment (time, cost, probability, availability, speciality,etc.) are respected.

Keywords : Planning, Scheduling, Temporal Constraints, AI, Incertainty and Agents.

Discipline : Informatique

Groupe de Recherche en Informatique, Image, Automatique et Instrumentation de Caen.
CNRS UMR 6072

Table des matières

Table des figures

Liste des tableaux

Chapitre 1

Introduction générale

1.1 Le contexte scientifique général

La planification est une sous-discipline de l'intelligence artificielle qui s'applique dans de nombreux problèmes tels que la robotique, la gestion de projets, la navigation sur Internet, la gestion des situations de crise, etc. L'objectif de la planification est de fournir à un système (robotique, informatique, humain,...) la capacité de raisonner pour interagir avec son environnement de façon autonome, afin d'atteindre les objectifs qui lui ont été assignés. Elle se définit en termes de problèmes à résoudre et se propose, étant donnés :

1. une représentation de l'état initial du monde,
2. un ensemble d'opérateurs de changement d'état du monde (qui représentent les actions qu'il est possible d'effectuer dans le monde),
3. un but à atteindre,

de donner les moyens à un système informatique de trouver une suite d'actions (c.-à-d. d'opérations directement exécutables) à appliquer sur le monde pour le faire passer de l'état initial à un état qui satisfait le but à atteindre. Un plan, élaboré par une partie du système appelée générateur de plan (ou planificateur), est un ensemble structuré d'actions décrivant les différents changements possibles du monde et dont l'exécution amène au but. La planification (ou génération de plan) est aussi le processus qui élabore cet ensemble d'actions. Un plan est correct si tous les buts désirés sont satisfaits suite à l'exécution des actions trouvées.

La difficulté du processus de planification dépend du degré de prise en compte des réalités de l'environnement. En effet, le monde réel étant évolutif, le plan engendré doit tenir compte de la nature dynamique et de l'imprévisibilité possible de ce dernier. Les approches les plus connues sont les suivantes :

1. Générer un plan sans tenir compte de la nature dynamique du monde. Cette technique consiste à considérer que les évolutions du monde sont prévisibles et que l'agent possède

toutes les connaissances à propos de l'environnement. Si lors de l'exécution du plan, un problème est survenu, il sera géré par une des deux méthodes appliquées dans ce cas : replanification complète ou réparation du plan.

2. Générer un plan en prenant en compte la connaissance incomplète du monde tout en considérant que les actions sont déterministes et ont des effets parfaitement connus.
3. Générer un plan en tenant compte de tout événement pouvant se produire. Cette approche considère que le monde est en mouvement continu et que tout est incertain. Elle est plus complexe que les deux précédentes, puisque l'agent doit effectuer des observations et des études à tout moment pour pouvoir suivre l'exécution du plan.

D'autres paramètres jouent aussi un rôle dans le degré de difficulté de la planification. Plus particulièrement :

- le temps : nombre de cas réels exigent le travail dans un délai bien précis ou bien le plus rapidement possible. Par exemple, le robot sur Mars doit prendre les photos avant le coucher du soleil ; plus encore, il peut être contraint d'exécuter une tâche pour pouvoir en exécuter une autre utilisant le résultat de la première.
- le coût : l'exécution de n'importe quelle activité a un coût que ce soit en argent, en matière première, en temps, en communication, etc.
- les ressources : la prise en compte des ressources en planification est un point important ; celle-ci est naturellement plus difficile si les ressources sont limitées. Planifier le trajet d'un robot est plus difficile si nous le supposons limité en énergie.

1.2 Le cadre de notre étude

Notre étude est consacrée au problème de la planification et de l'ordonnancement des tâches sous contraintes temporelles et incertitude. Les contraintes temporelles que nous traitons dans cette thèse sont de deux types : (1) les contraintes temporelles qualitatives (ou symboliques), c'est-à-dire la précédence entre les tâches, divisées elles mêmes en deux types (contraintes conjonctives et contraintes disjonctives) et (2) les contraintes temporelles quantitatives (ou numériques), c'est-à-dire des données temporelles relatives aux tâches comme la date de début, la date de fin et les durées d'exécution. Dans cette étude, l'incertitude sur la durée des tâches se traduit par une distribution de probabilités sur un ensemble fini.

Étant donnés une description de l'état initial du monde, un ensemble d'opérateurs ou d'actions possibles et un but à atteindre, notre objectif est de trouver une suite d'actions exécutables à appliquer sur le monde pour le faire évaluer de façon que le but soit atteint tout en satisfaisant les contraintes du domaine.

Une particularité importante du problème considéré est que les tâches et les contraintes sont représentées à l'aide d'un graphe ET/OU et que les durées des tâches sont pondérées par

des probabilités d'exécution. Celles-ci expriment une incertitude sur la connaissance exacte des durées d'exécution des tâches qui ne seront réellement connues que lors de l'exécution effective. Ainsi, une tâche s'exécute durant l'une de ses durées d'exécution possibles avec la probabilité associée à celle-ci. Les diverses tâches sont représentées dans un graphe ET/OU où les nœuds représentent les tâches et les arcs quant à eux représentent les relations de précédence entre les tâches. Étant donné ce graphe, notre objectif est de déterminer un plan de tâches qui satisfait toutes les contraintes et qui répond aux critères de choix exigés par l'utilisateur en terme de temps, coût et probabilité.

Nous générons, tout d'abord, selon les contraintes de précédence entre les tâches en utilisant la recherche en chaînage arrière dans le graphe ET/OU, tous les plans candidats à être un plan-solution[1]. Ensuite, parmi eux, nous sélectionnons ceux qui satisfont les contraintes temporelles quantitatives sur les tâches. Nous utilisons pour cela les techniques de propagation de contraintes temporelles proposées dans [Bresina & Washington 00] mais adaptées à notre problème. Enfin, nous sélectionnons le plan-solution et l'ordonnancement-solution selon le ou les critères de préférence de l'utilisateur. L'ordonnancement-solution est une suite de tâches devant s'exécuter durant des intervalles d'exécution bien définis. Afin de choisir l'ordonnancement-solution, nous calculons l'union d'intervalles pour un ensemble d'intervalles d'exécution et ensuite nous choisissons celui qui répond le mieux au critère de sélection.

1.2.1 Les hypothèses

Nous travaillons dans un système de planification contenant un seul agent qui a comme but l'élaboration d'un plan composé d'une suite de tâches exécutables. Cet agent, *"le planificateur"*, analyse, choisit, gère et ordonne un ensemble des tâches reliées entre elles par des contraintes de précédence et possédant chacune un ensemble de contraintes temporelles quantitatives (une date de début, une date de fin et des durées d'exécution). Le processus de planification est élaboré hors-ligne, c'est-à-dire que le planificateur prépare tout d'abord un plan-solution complet et commence ensuite son exécution. Nous ne nous sommes intéressés dans cette thèse qu'à la "planification des tâches" dans un environnement incertain et sous contraintes temporelles et de coût. Nous faisons aussi l'hypothèse que les ressources demandées pour l'exécution des tâches sont toujours disponibles et illimitées, ce qui n'est pas souvent vrai dans le monde réel : cette hypothèse a pour but de se concentrer particulièrement sur la planification temporelle et probabiliste, sans tenir compte d'autres ressources.

[1] Le plan-solution est celui sélectionné pour être exécuté.

1.2.2 Cas d'application

Nous avons appliqué notre méthode de planification à un cas pratique relativement complexe qui concerne la planification d'un ensemble d'agents travaillant ensemble dans un lieu afin d'atteindre un but donné tout en respectant les délais et les contraintes. À partir de l'ensemble des données concernant les agents et les buts, nous construisons un graphe ET/OU où nous appliquons ensuite notre méthode de planification. Le but de cette application est de montrer l'efficacité de notre approche sur un exemple réel.
Les tests expérimentaux ont montré la performance de notre système de planification. Plus particulièrement, notre planificateur est correct, sain et complet [Régnier 90] :

- un plan est correct si et seulement si l'exécution de ce plan permet, partant de l'état initial du monde, d'aboutir à un état but ;
- un planificateur est sain si et seulement si tous les plans qu'il peut produire sont corrects c.-à-d. que l'exécution de ces plans permet, partant de l'état initial du monde, d'aboutir à un état but ;
- un planificateur est complet si et seulement s'il produit un plan permettant de résoudre le problème lorsqu'un tel plan existe.

1.3 L'organisation du rapport

Le plan de ce mémoire est composé de trois parties. La première partie est consacrée à l'état de l'art. Après une introduction sur les méthodes de l'ordonnancement les plus connues, nous détaillons dans le chapitre 2, les techniques utilisées dans les planificateurs classiques. Ces derniers sont fondés sur des hypothèses simplificatrices comme la stabilité du monde, la certitude des actions et l'observabilité totale de l'environnement. Nous étudions aussi les différentes méthodes de recherche utilisées dans la littérature : CSP, la planification hiérarchique et les systèmes de planification prenant en compte l'exécution.
Dans le chapitre 3, nous présentons quelques planificateurs temporels. Certains d'entre eux prennent en compte les contraintes temporelles qualitatives, d'autres les contraintes temporelles quantitatives, et peu combinent les deux ensemble. Nous étudions les différentes sortes possibles d'utilisation du temps dans la planification et nous faisons une synthèse sur les limites de tels planificateurs.
Dans le chapitre 4, nous étudions les planificateurs opérant sous incertitude. Ces planificateurs utilisent l'incertitude sous forme conditionnelle, probabiliste, conditionnelle probabiliste ou bien décisionnelle. Quelques planificateurs combinent le temps et l'incertitude. Ces derniers sont présentés dans la section 4.4 du même chapitre. Cette partie s'achève par une synthèse sur les planificateurs les plus connus dans la littérature et ainsi que l'introduction de notre approche via

une étude comparative de celle-ci avec celles déjà existantes.

La deuxième partie de cette thèse est consacrée à notre contribution. Après une brève introduction de la problématique motivant notre recherche et les hypothèses posées, nous détaillons dans le chapitre 5 les deux premières étapes de notre méthode de planification. Il s'agit de la génération des plans *faisables*, c.-à-d. les plans qui satisfont les contraintes de précédence entre les tâches du graphe ET/OU et celle des plans *admissibles*. Ces derniers satisfont les contraintes temporelles locales des tâches des plans faisables. Ils sont obtenus suite à la propagation des durées sur les tâches afin de calculer les intervalles d'exécution possibles. Ensuite, nous vérifions l'admissibilité des plans faisables en comparant les intervalles d'exécution trouvés et les fenêtres temporelles initiales des tâches. Dans le chapitre 6, nous calculons les valeurs espérées de temps et de coût par intervalle d'exécution, par tâche et par plan en propageant les probabilités et les coûts sur l'ensemble des tâches des plans admissibles. Le chapitre 7 est consacré à la sélection du plan-solution et de l'ordonnancement-solution selon les critères de préférence de l'utilisateur. Le chapitre 8 est consacré à un exemple d'application. Il s'agit de la gestion d'une situation de crise où plusieurs agents doivent collaborer sans communiquer afin d'atteindre les buts désirés tout en satisfaisant toutes les contraintes et respectant les données du domaine comme la disponibilité des agents, leur capacité, etc. Après avoir décrit le problème, nous présentons la solution adoptée : il s'agit de la transformation du problème en graphe ET/OU et l'application de notre algorithme de planification sur le graphe obtenu. Le plan-solution trouvé est ensuite transformé en commandes compréhensibles par les agents concernés par son exécution.
Les tests et les analyses sont présentés dans le chapitre 9. Nous montrons que notre système de planification est efficace pour résoudre des problèmes de planification temporelle et probabiliste. Il possède une expressivité assez riche et une complexité raisonnable pour traiter un exemple réel de planification.

La troisième partie contient un seul chapitre. Il s'agit de la conclusion et d'un ensemble d'idées de poursuite de cette recherche.

Première partie

Formalismes et approches

Dans cette partie, nous présentons les principales méthodes de planification étudiées en Intelligence Artificielle. Nous consacrons, d'abord, le chapitre 2 à une petite introduction sur la planification et l'ordonnancement, ensuite nous présentons brièvement les méthodes d'ordonnancement les plus connues, les planificateurs classiques, les méthodes de recherche les plus utilisées, la planification CSP, la planification hiérarchique et les planificateurs qui contrôlent l'exécution. Dans le chapitre 3, nous présentons les planificateurs qui prennent en compte le temps. Le chapitre 4 est consacré aux planificateurs prenant en compte l'incertitude, que ce soit l'incertitude conditionnelle, l'incertitude probabiliste, l'incertitude conditionnelle probabiliste, l'incertitude temporelle probabiliste ou le processus décisionnel de Markov. Nous terminons ce chapitre par l'introduction de notre propre approche de la planification. Un schéma vaut mieux qu'un long discours. Nous synthétisons dans la figure 1.1 l'organisation de différents types d'ordonnancement et de planification étudiés dans cette partie.

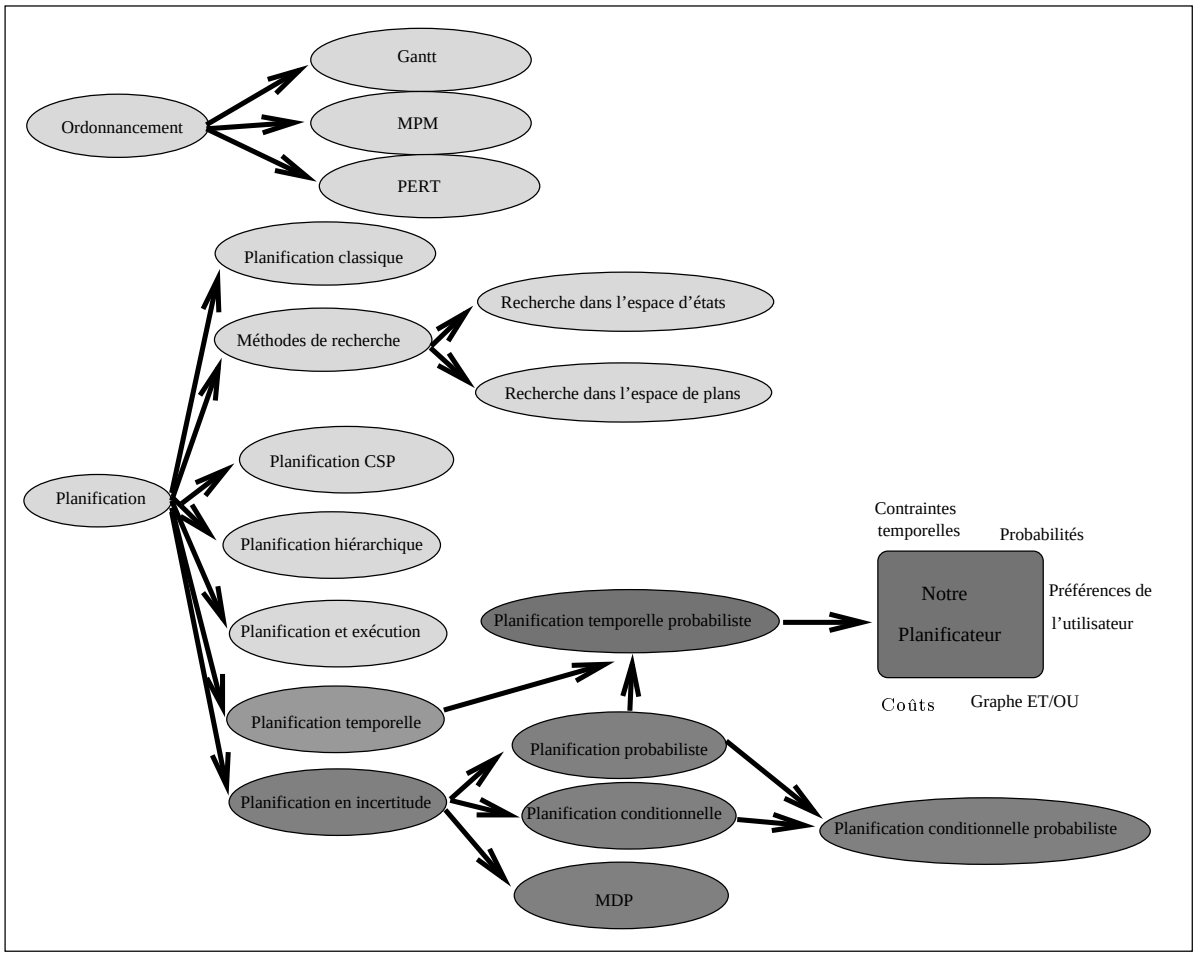

FIG. 1.1 – Schéma représentant les différents types de planification introduits dans cette partie. Les couleurs, de plus claires aux plus foncées, indiquent la succession des chapitres

Chapitre 2

Introduction au domaine de la planification

2.1 Préambule

La gestion d'un projet se déroule en quatre étapes :

1. **La planification :** qui vise à déterminer les différentes opérations à réaliser et les moyens matériels et humains à y affecter.
2. **L'ordonnancement :** qui vise à déterminer les différentes dates correspondant aux activités.
3. **L'exécution :** qui consiste à la mise en œuvre des différentes opérations définies dans la phase d'ordonnancement.
4. **Le contrôle :** qui consiste à superviser l'exécution et voir si celle-ci respecte les prévisions.

Dans cette thèse, notre travail se concentre sur les deux premières étapes : la planification et l'ordonnancement.

Le problème de la planification et de l'ordonnancement des tâches représente l'un des problèmes les plus complexes dans le domaine de l'Intelligence Artificielle. Parmi les situations les plus connues, citons la gestion de crise, la gestion de production, la gestion de projet, la robotique, le domaine médical,...

La planification, un élément indispensable dans l'élaboration de systèmes intelligents, se définit en termes d'un domaine de planification et de problèmes à résoudre. Un domaine de planification est décrit par un ensemble d'actions qui vont permettre des transitions entre les états. Un problème de planification consiste en une description de l'état initial et de l'état but. Plus formellement, étant donnés un ensemble d'opérations $O = \{o_1, o_2, ..., o_n\}$, un état initial $I = s_0$ et un état final à atteindre $G = s_g$, un problème de planification est défini par $\Pi = (I, G)$. La

solution de ce problème est un plan complet $P = \{a_0, a_1, ..., a_g\}$ décrit par un ensemble d'actions a_i qui vont permettre des transitions de l'état initial à l'état final.

L'ordonnancement consiste à organiser dans le temps un ensemble d'activités de façon à satisfaire un ensemble de contraintes et optimiser le résultat. Les techniques d'ordonnancement ont pour objectif de répondre au mieux aux besoins exprimés par un client, au meilleur coût et dans les meilleurs délais, en tenant compte des différentes contraintes.
Nous introduisons dans les deux sections suivantes les définitions de la planification et de l'ordonnancement et dans la section 2.4, la différence entre les deux techniques.

2.2 La planification

La planification en Intelligence Artificielle [Ghallab *et al.* 04] concerne la résolution des problèmes dans plusieurs domaines. Les solutions de ces problèmes sont sous la forme d'une suite d'opérations ou d'actions à réaliser dans le but d'atteindre l'objectif. Une telle solution est appelée un *plan*. Le programme qui est capable de trouver un plan est un *planificateur*.

La figure 2.1 représente un planificateur qui prend en entrée un problème et un domaine de planification et fournit en sortie un plan.

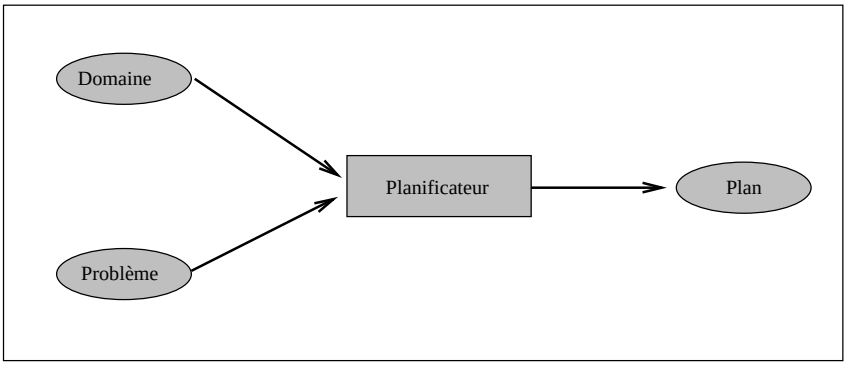

Fig. 2.1 – Un planificateur prend en entrée un problème et un domaine de planification et produit en sortie un plan

L'objectif de la planification est de fournir à un système (robot, ordinateur, machine,...) la capacité de raisonner pour atteindre un but qui lui a été attribué. On peut distinguer deux types de planification selon le niveau d'abstraction auquel on se situe. Le bas niveau représente une tâche d'un plan. Par exemple, demander à un robot d'effectuer un déplacement d'un lieu (L_1) à un autre (L_2) est une tâche d'un plan global qui serait "se déplacer de L_1 à L_2 et prendre des photos". Afin de faire le trajet de L_1 à L_2, le robot doit tenir compte de la nature du sol, des obstacles, des ressources, etc. Le haut niveau représente un niveau plus abstrait dans le sens où on ne prend pas en compte les propriétés physiques des informations. Par exemple, "sortir

d'une navette, se déplacer de L_1 à L_2, prendre des photos, se déplacer de L_2 à L_1, monter dans la navette" est un plan du haut niveau. Le but global essentiel du robot est de faire les tâches "sortir", "se déplacer", "prendre des photos", "revenir à sa place" et "monter" sans se soucier de la façon de les faire. Un plan est correct si tous les buts désirés sont satisfaits en fonction de l'état initial et des opérations trouvées. La planification doit répondre à la question : quelles tâches l'agent doit-il effectuer et dans quel ordre ? La difficulté de la planification dépend du degré de la prise en compte de la réalité.

2.3 L'ordonnancement

Historiquement, les problèmes d'ordonnancement ont été initialement abordés dans le domaine de la recherche opérationnelle (théorie des graphes, programmation dynamique, programmation linéaire, méthodes d'optimisation combinatoire), mais ont vite montré leurs limites en termes d'expressivité. L'intelligence artificielle s'est alors penchée sur le problème, renouvelant les techniques employées grâce à une représentation plus riche des connaissances du domaine (problèmes de satisfaction de contraintes, algorithmes de propagation de contraintes, langages de programmation par contraintes). Un problème d'ordonnancement se pose lorsqu'il s'agit d'organiser dans le temps l'exécution d'un ensemble de tâches. De tels problèmes se rencontrent dans différents contextes tels que la gestion de grands projets, la conduite d'ateliers de fabrication, l'organisation d'activités de service, ...

La résolution d'un problème d'ordonnancement [Esquirol & Lopez 99, Lopez & Roubellat 01] consiste à placer dans le temps des activités ou tâches, compte tenu de contraintes temporelle (délais, contraintes de précédence,...) et de contraintes portant sur l'utilisation et la disponibilité des ressources requises par les tâches. Un ordonnancement décrit l'exécution des tâches en précisant leurs dates de début et leurs dates de fin et l'allocation des ressources au cours du temps, et vise à satisfaire un ou plusieurs objectifs. Les problèmes d'ordonnancement que nous considérons dans cette thèse ne sont soumis qu'à des contraintes temporelles. Nous ne traitons pas les contraintes de ressources (nous supposons que celles-ci sont illimitées et disponibles à tout moment).

2.4 Planification et ordonnancement

La planification raisonne par rapport au but fixé et cherche à déterminer les tâches qui vont permettre de résoudre ce but. L'ordonnancement part au contraire d'un ensemble de tâches connues à l'avance, qu'il s'agit de positionner dans le temps les unes par rapport aux autres. On peut considérer que l'ordonnancement est un aspect particulier du problème de planification. En fait, planification et ordonnancement tendent actuellement à se rejoindre, le premier domaine

s'enrichissant par la possibilité de placer les tâches les unes par rapport aux autres en fonction de contraintes temporelles (et de ressources s'il y en a), le second quant à lui souhaitant désormais gérer des choix entre tâches équivalentes.

Dans cette thèse, nous considérons que la planification et l'ordonnancement servent à produire un plan complet, partant d'un état initial pour atteindre un état final, en respectant toutes les contraintes existantes dans le domaine de planification et d'ordonnancement. Puisque planifier et ordonner ne sont pas deux étapes distinctes dans notre approche, et pour simplifier, nous utilisons le terme "planification" pour désigner "planification et ordonnancement". Notre planificateur est en pratique un système de planification et d'ordonnancement.

Dans ce chapitre, nous allons présenter un ensemble de planificateurs qui ont eu un certain succès dans la résolution de problèmes de planification. Nous commençons dans la section 2.5 par une présentation de quelques systèmes d'ordonnancement. Ensuite, nous développons quelques systèmes existants dans le domaine de la planification classique.

2.5 Les méthodes d'ordonnancement

Les méthodes d'ordonnancement les plus connues sont : le diagramme de Gantt, la méthode MPM[2] et le PERT[3]. Nous résumons dans les sections suivantes ces trois techniques d'ordonnancement.

2.5.1 Le diagramme de Gantt

Le diagramme de Gantt [Gantt] est un outil permettant de modéliser la planification de tâches nécessaires à la réalisation d'un projet. Le principe de ce type de diagramme est de représenter au sein d'un tableau, en ligne les différentes tâches et en colonne les unités de temps (exprimées en mois, semaines, jours,...). Les différentes étapes de réalisation d'un diagramme de Gantt sont les suivantes :

1. Définition des différentes tâches à réaliser et leurs durées.
2. Définition des relations d'antériorité entre tâches.
3. Représentation des tâches par des traits dans le diagramme : d'abord les tâches n'ayant aucune antériorité, puis celles dont les tâches antérieures ont déjà été représentées, et ainsi de suite...
4. Représentation de la progression réelle du travail par un trait pointillé parallèle à la tâche planifiée.

[2] Méthode des Potentiels Métra.
[3] Program Evaluation and Review Technic.

La figure 2.2 représente un diagramme de Gantt où chaque colonne représente une unité de temps, les traits épais représentent les durées d'exécution prévues des tâches et les traits pointillés représentent le déroulement d'exécution. Par exemple, la tâche B, qui dure 5 unités de temps, ne peut commencer son exécution qu'après la fin de la tâche A et elle peut s'exécuter en même temps que la tâche C.

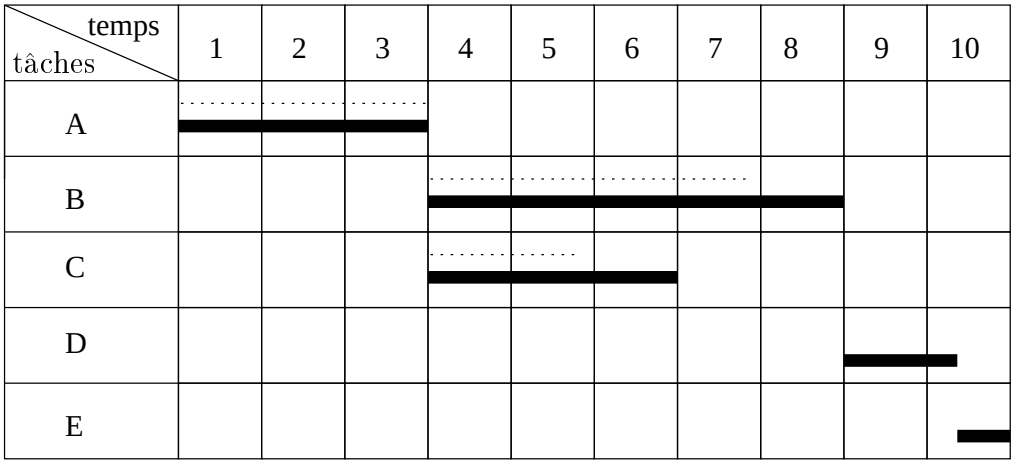

FIG. 2.2 – Un exemple du diagramme de Gantt

Le chemin critique est formé d'une succession de tâches sur le chemin le plus long en terme de durées (A,B,D,E dans l'exemple). Il est appelé chemin critique parce que tout retard pris sur l'une des tâches de ce chemin entraîne du retard dans l'achèvement du projet. Le diagramme de Gantt permet de déterminer la date de réalisation d'un projet et d'identifier les marges existantes sur certaines tâches (avec une date de début au plus tôt et une date de fin au plus tard). Son point faible est que son application est limitée à des problèmes particuliers.

2.5.2 MPM

Dans la méthode des potentiels-métra [ROY & DIBON 66], le problème est représenté sous forme d'un graphe tel que les tâches sont représentées par des nœuds et les contraintes de succession par des arcs. À chaque nœud sont associées une date de début au plus tôt et une date de fin au plus tard. À chaque arc est associé un délai d'attente entre les tâches. La figure 2.3 représente un exemple de la méthode des potentiels-métra. La date de début au plus tôt d'une tâche dépend de la date de fin des tâches qui la précèdent. La tâche DEBUT est initialisée avec une date de début au plus tôt égale à zéro. Cette méthode permet de déterminer la date de réalisation d'un projet ainsi que la date de début et de fin de chaque tâche mais elle est incapable de résoudre des problèmes qui prennent en compte plus de contraintes telles que l'incertitude et les coûts d'exécution des tâches.

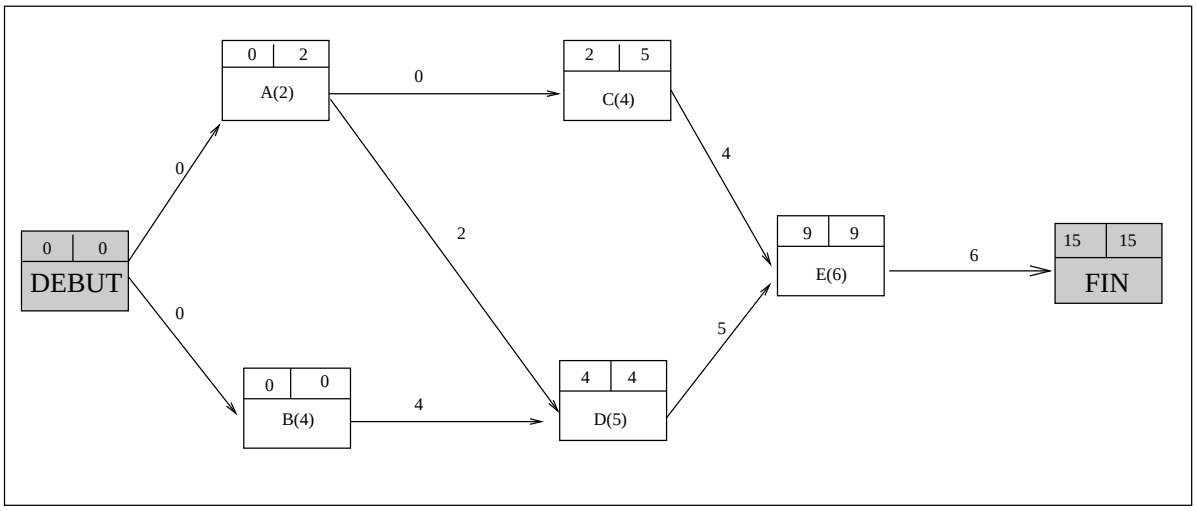

FIG. 2.3 – Un exemple d'un graphe MPM

2.5.3 PERT

La méthode PERT[4] [ALLEN & HAMILTON 64, AIKAT 96] consiste à mettre en ordre sous forme de réseau plusieurs tâches qui grâce à leurs dépendances et à leur chronologie permettent d'avoir un produit fini. Elle représente le problème sous forme d'un graphe tel que les tâches sont représentées par un arc auquel on associe un nombre entre parenthèses qui représente la durée de la tâche. Un nœud représente la fin d'une ou de plusieurs tâches. La figure 2.4 représente un graphe PERT. La construction d'un tel graphe se fait par niveau : le niveau 1 contient les tâches sans antécédents, le niveau i contient les tâches dont les antécédents sont exclusivement du niveau $i-1$. La date de début au plus tôt d'une tâche dépend des dates de fin des tâches qui la précèdent. Cette méthode permet de déterminer la date de début et de fin de chaque tâche ainsi que le chemin critique c'est-à-dire un ensemble d'activités tel que tout retard dans leur exécution provoquerait un retard de la fin du projet. Par contre, elle ne présente pas d'échelle calendaire comme dans le cas de diagramme de Gantt.

2.6 La planification classique

La planification cherche à concevoir des systèmes capables de générer automatiquement un résultat sous la forme d'un système intégré de décisions appelé plan-solution. Ce dernier est une collection organisée de descriptions d'opérations ; il est destiné à guider l'action d'un ou plusieurs agents exécuteurs (systèmes robotiques ou humains) qui doivent agir dans un monde particulier pour atteindre un but préalablement défini. L'exécution est la réalisation d'actions effectuée

[4]Program Evaluation Review Technique.

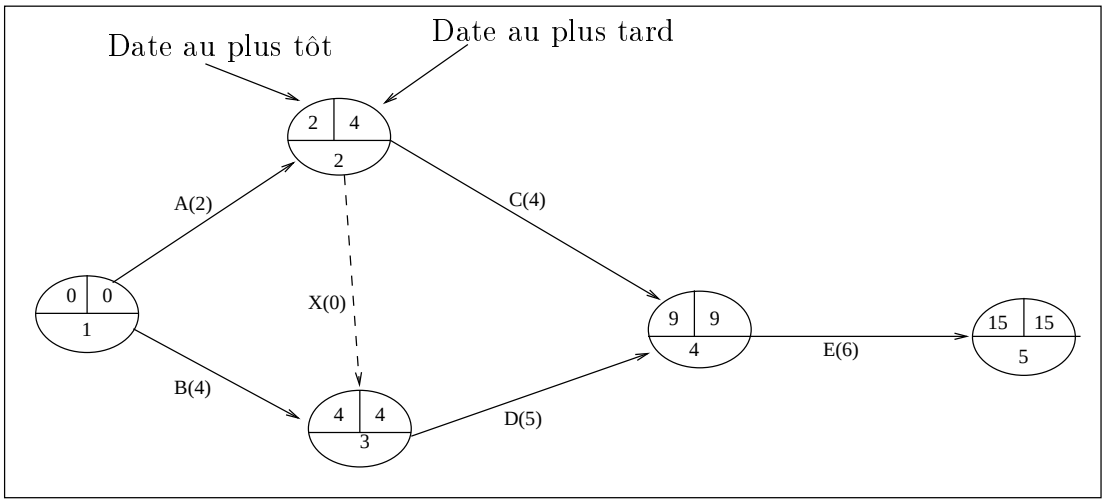

FIG. 2.4 – Un exemple d'un graphe PERT

suivant les directives du plan. Elle vise à réaliser la prédiction que constitue ce plan ; lorsqu'elle est conforme à ce qu'il indique, elle doit permettre (en l'absence d'événements imprévus et si la modélisation du monde est pertinente) de faire évoluer l'univers de l'état initial vers un état satisfaisant le but.

L'apparition de la planification en IA a conduit aux planificateurs dits classiques que nous présentons dans les sections suivantes. Deux hypothèses sont la plupart du temps retenues dans ce type de planificateurs. Il s'agit de la complétude de la description du monde et du déterminisme des actions. La première hypothèse s'attache au fait que le planificateur doit avoir toute la connaissance disponible pour trouver une solution correcte. Une action est déterministe si ses effets sont totalement prévisibles sachant le contexte dans lequel elle s'exécute.

2.6.1 STRIPS

STRIPS[5] [FIKES & NILSSON 71], apparu en 1970, est un des premiers planificateurs dans le domaine de l'Intelligence Artificielle. Il définit les actions sous forme d'opérateurs, chacun est composé des *pré-conditions* (propriétés) nécessaires à l'exécution de l'action et des *effets* engendrés par sa mise en application. Les pré-conditions et les effets forment une *proposition* qui peut être vraie ou fausse. Un effet, sous la forme *ajouter à la liste* ou *supprimer de la liste*, a comme objectif de rendre la proposition vraie. La partie droite de la figure 2.5 représente un opérateur STRIPS composé de ses pré-conditions, de ses effets et de son action. Considérons par exemple, un robot qui va ouvrir une porte. *porte ouverte* et *porte fermée* sont deux propositions telles que, à un moment donné, l'une est vraie et l'autre est fausse. L'action *ouvrir la porte* doit

[5] STandford Research Institue Problem Solver.

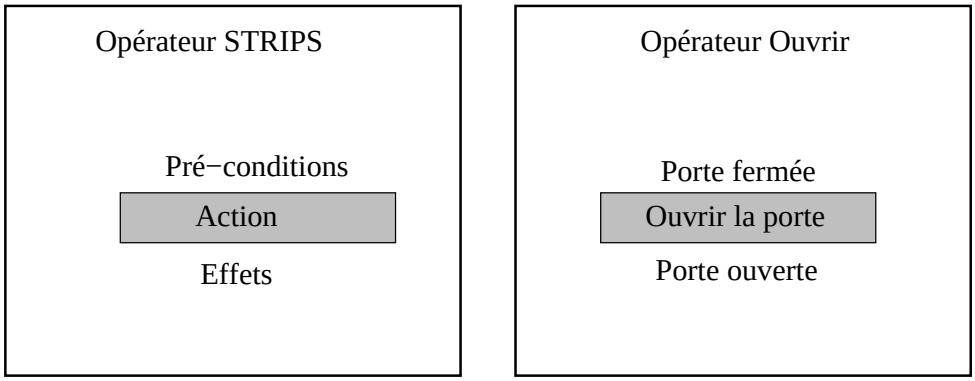

FIG. 2.5 – Exemple d'opérateurs STRIPS

avoir la proposition *porte fermée* comme une pré-condition et comme un effet *supprimer de la liste* et la proposition *porte ouverte* comme un effet *ajouter à la liste*. La partie gauche de la figure 2.5 illustre l'opérateur *ouvrir*. La terminologie de STRIPS est très intéressante dans la mesure où elle est simple et permet de modéliser beaucoup de problèmes de planification, ce qui prouve son utilisation dans plusieurs formalismes en particulier le langage ADL[6] [PEDNAULT 89] en ajoutant des pré-conditions négatives, des effets conditionnels, des variables, etc. Par contre, dans ce planificateur, démontré PSPACE-complet, toutes les actions sont déterministes ; c'est à dire que tous les effets de l'action sont connus et bien déterminés et qu'une action ne peut se déclencher que dans un certain contexte. L'algorithme général de STRIPS est représenté dans l'algorithme 1 et est du type suivant : le démonstrateur cherche à prouver que l'union de l'état initial E_0 et de la négation du but B_0 est inconsistante (donc B_0 est vérifié dans l'état E_0).

Depuis ce système fondateur, et dans le cadre "classique" de la planification qu'il a délimité (environnement statique, observabilité totale, agent omniscient, actions atomiques et déterministes), de très gros progrès ont été réalisés. En particulier le planificateur GraphPlan (section 2.6.2), apparu en 1995, fut à l'origine d'un bouleversement dans le domaine de la planification classique. Ses performances et les idées qui guidaient sa conception ont inspiré un grand nombre de travaux.

2.6.2 GraphPlan

GraphPlan [BLUM & FURST 97] est l'un des planificateurs classiques les plus connus grâce à ses performances le distinguant par rapport à tous ses prédécesseurs. Son apparition a donné une évolution importante aux recherches dans le domaine de la planification en IA. Celles-ci ont permis aux algorithmes de planification STRIPS d'augmenter leurs performances de manière si importante que l'on peut maintenant envisager des applications réelles à moyen terme. Il

[6] Action Description Language.

Données :
- Pile des buts $\leftarrow B_0$;
- État courant $\leftarrow E_0$;
- *Succès* $\leftarrow faux$;
- *Échec* $\leftarrow faux$;

début

tant que *non (Succès ou Échec)* **faire**

1. sélectionner un sous-but *Premier point de retour arrière* et essayer d'établir qu'il est vérifié dans le modèle du monde associé **si** *cela est vrai* **alors** aller en 4 ;
2. **si** *c'est possible* **alors** choisir un opérateur approprié (opérateur dont l'effet "ajouter à la liste" comporte des clauses qui permettent de poursuivre la démonstration du 1) *Second retour arrière* **sinon** *Échec* $\leftarrow Vrai$;
3. empiler comme nouveaux sous-buts, les pré-conditions instanciées de l'opérateur sélectionné et aller en 1 ;
4. **si** *le sous-but est le but initial* **alors** *Succès* $\leftarrow Vrai$; **sinon** créer un nouveau modèle du monde en appliquant l'opérateur dont les pré-conditions constituent le sous-but qui vient d'être établi, déplier ce sous-but et aller en 1 ;

fin

fin

Algorithme 1 – L'algorithme général de STRIPS

s'agit d'un générateur de plans complets, qui utilise une description du problème basée sur le formalisme STRIPS, et autorise une certaine parallélisation des actions. La contribution la plus importante de GraphPlan est une structure compacte appelée graphe de planification, qui entrelace des niveaux de nœuds de propositions et d'actions[7] construits en chaînage avant, plus des exclusions mutuelles qui représentent l'impossibilité pour deux actions d'être exécutées en parallèle, ou pour deux propositions d'être présentes dans un même état après l'application d'un plan parallèle d'une longueur donnée. Cette structure est ensuite explorée par un algorithme de recherche en chaînage arrière. Le graphe de planification est ensuite étendu niveau par niveau jusqu'à ce qu'une solution soit trouvée ou qu'une condition d'arrêt soit vérifiée. Le graphe de planification peut aussi être traduit en une formule booléenne dont les modèles correspondent aux plans solutions. La taille et le temps de création de ce graphe sont polynomiaux par rapport à la taille du problème. Un plan en GraphPlan est une séquence d'ensembles d'actions. Chacun de ces ensembles contient des actions qui doivent être exécutées simultanément selon l'ordre de l'ensemble dans le plan.

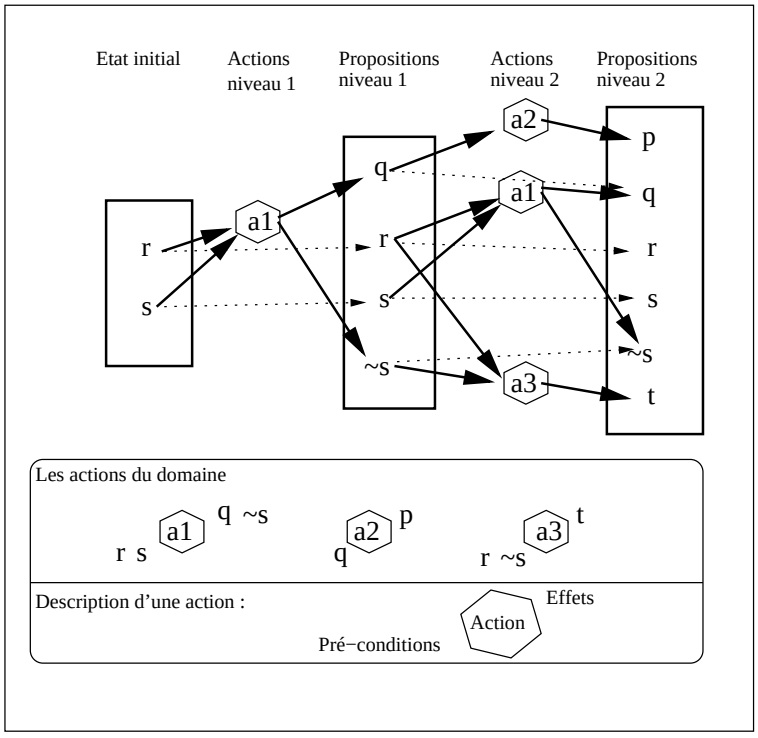

FIG. 2.6 – Un exemple d'un graphe de planification en GraphPlan

Deux actions $a1$ et $a2$ sont mutuellement exclusives dans trois cas :

– Interférence : un effet de $a1$ est la négation d'une pré-condition de $a2$.

[7] Il existe un arc d'un nœud de proposition à un nœud d'action pour chaque pré-condition d'une action et un arc d'un nœud d'action à un nœud de proposition pour chacun de ses effets.

- Effets inconsistants : un effet de $a1$ est la négation d'un effet de $a2$.
- Compétition : $a1$ et $a2$ ont des pré-conditions mutuellement exclusives.

Deux pré-conditions $P1$ et $P2$ sont mutuellement exclusives dans deux cas :

- Une pré-condition est la négation d'une autre ($P1 = \sim P2$).
- Toutes les actions pouvant supporter $P1$ sont mutuellement exclusives avec toutes celles pouvant supporter $P2$.

La figure 2.6 représente un graphe de planification composé de l'état initial et deux niveaux. Les propositions (pré-conditions et effets) sont représentées par des lettres (p,q,...) et les actions par $a1$, $a2$ et $a3$. Par exemple, r et s sont les pré-conditions de l'action $a1$, q est son effet *"ajouter à la liste"* et $\sim s$[8] est son effet *"supprimer de la liste"*. Les pré-conditions de l'action $a3$ sont r et l'absence de s et son effet *"ajouter à la liste"* est t. L'algorithme de recherche des plans en GraphPlan est composé de deux grandes phases :

1. **La construction d'un graphe de planification :**
   Chaque niveau A_i contient toutes les actions qui peuvent être appliquées en S_i, avec des contraintes indiquant quelles paires d'actions ne peuvent pas être exécutées en même temps. Chaque niveau S_i contient tous les effets qui peuvent être le résultat d'une action en A_{i-1}, avec des contraintes indiquant les paires impossibles.
   La fonction de la construction d'un graphe de planification est représenté dans l'algorithme 2.

Fonction EXTRAIRE-GRAPHE
début

- Les propositions de niveau initial
  seulement les conditions initiales
- Les propositions du niveau i
  si *toutes les pré-conditions d'une action sont dans i-1* **alors** Ajouter action au niveau i
- Les propositions du niveau i+1
  pour chaque *action au niveau i* **faire**
  Ajouter tous ses effets au niveau i+1
  Permettre aux propositions du niveau i de persister au niveau i+1
  fin
  Terminer quand aucune nouvelle proposition non-exclusive mutuellement peut être ajouter

fin

Algorithme 2 – L'algorithme de la construction d'un graphe de planification dans GraphPlan

[8] La proposition $\sim s$ est la négation de la proposition s.

2. **L'extraction d'une solution :**
   - Recherche en arrière, avec comme état initial le niveau S_n.
   - À chaque niveau S_i, choisir toutes les actions sans conflits dans A_{i-1} qui accomplissent les buts.
     - Sans conflits veut dire qu'il n'y a pas de liens mutuellement exclusifs entre les actions choisies ou entre leurs pré-conditions.
     - Le niveau S_{i-1} a comme buts les pré-conditions des actions choisies en A_{i-1}.
   - Le but est d'atteindre l'état S_0 avec tous les buts accomplis.

   La fonction de l'extraction d'une solution en GraphPlan est représenté dans l'algorithme 3. L'algorithme de recherche des plans en GraphPlan qui utilise les deux fonctions EXTRAIRE-GRAPHE et EXTRAIRE-SOLUTION est représenté dans l'algorithme 4.

```
Fonction EXTRAIRE-SOLUTION(graphe,g,j)
début
    si j=0 alors une solution est trouvée
    pour chaque proposition p dans g faire
        choisir une action pour l'état S_{j-1} pour achever p
    fin
    si aucune paire des actions choisies est mutuellement exclusive alors retour en
    arrière
    g' : {les pré-conditions des actions choisies}
    EXTRAIRE-SOLUTION(graphe,g',j)
fin
```

Algorithme 3 – L'algorithme de l'extraction d'une solution dans GraphPlan. g représente l'ensemble de buts et j représente le niveau de l'état S_j

De tous les planificateurs de sa catégorie, GraphPlan s'est révélé être le plus performant selon les critères suivants :

- taille des problèmes traités,
- rapidité de résolution des problèmes,
- optimalité des solutions proposées.

L'utilisation de la méthode de recherche dans l'espace de graphe de planification permet de générer des plans optimaux en nombre de niveaux et possédant une bonne flexibilité. Celle-ci reste généralement moins bonne que celle obtenue par recherche dans les espaces de plans partiels (cf. section 2.7.2). Par contre, cette approche évite une partie de la redondance de la recherche dans l'espace des états (cf. section 2.7.1). La qualité de l'approximation dépend des ensembles mutuellement exclusifs pris en compte, ce qui entraîne une construction plus longue du graphe

```
Fonction GRAPHPLAN(problème)
Données : graphe ← graphe_initial(problème)
          buts ← BUTS(problème)
Résultat : solution ou échec
début
    si tous les buts ne sont pas exclusive mutuellement en dernier niveau du graphe alors
        solution ← EXTRAIRE-SOLUTION(graphe, goals, taille(graphe))
        si solution <> échec alors retourne solution
        sinon si non_solution_possible(graphe) alors retourne échec
        graphe ← EXTRAIRE-GRAPHE(graphe, problème)
    fin
fin
```

Algorithme 4 – L'algorithme de recherche des plans en GraphPlan ; les fonctions EXTRAIRE-GRAPHE et EXTRAIRE-SOLUTION sont les deux phases écrites dans les algorithmes 2 et 3.

(calcul des ensembles mutuellement exclusifs) et une extraction moins longue de la solution (moins d'actions à considérer, moins de retours en arrière). Une des limitations de GraphPlan est qu'il est applicable seulement aux domaines basés sur le formalisme STRIPS. En particulier, les actions ne peuvent pas créer des nouveaux objets et les effets des actions sont déterministes. Finalement, puisque GraphPlan garantit de trouver le plan le plus court possible, il arrive que des problèmes deviennent plus compliqués que dans la réalité. En effet, le compromis entre la qualité du plan trouvé et le temps de planification est une question ouverte dans la plupart des planificateurs.

GraphPlan a inspiré toute une génération de planificateurs : des améliorations et des innovations ont été faites, et son utilisation comme heuristique est à la base de plusieurs approches comme le montre la section suivante.

2.6.3 Planificateurs basés sur GraphPlan

Plusieurs techniques ont été employées pour améliorer GraphPlan : la réduction de l'espace de recherche avant la phase d'extraction du plan [Fox & Long 98, Nebel *et al.* 97], l'amélioration du langage de représentation des domaines et des problèmes [Koehler *et al.* 97, Weld *et al.* 98, Guere & Alami 99, Koehler 98] et l'amélioration de la phase d'extraction de la solution [Kambhampati 99, Kautz & Selman 99, Long & Fox 99, Fox & Long 99, Zimmerman & Kambhampati 99]. Dans cette section, nous présentons les planificateurs IPP, LCGP et AltAlt ; tous les trois sont basés sur les techniques de GraphPlan.

– Une technique permettant de réduire la taille du graphe de planification est proposée dans [Nebel *et al.* 97], pour le planificateur IPP fondé sur GraphPlan. Comme dans GraphPlan, la construction du graphe de planification se fait de l'état initial à l'état final et la recherche d'un plan solution se fait par la méthode de recherche en arrière. Elle offre plusieurs heuristiques qui permettent de supprimer de l'état initial les effets qui ne sont probablement pas utiles pour résoudre le problème. L'inconvénient majeur est donc qu'elle rend GraphPlan incomplet, mais il semble qu'elle fournisse pourtant de bons résultats. De plus, il y a de fortes chances pour que GraphPlan échoue rapidement si l'heuristique a supprimé trop d'effets ; il reste donc toujours la possibilité de relancer la planification avec l'état initial complet. Le principe de ce calcul est de construire un graphe ET/OU à partir des buts du problème. La racine de l'arbre est un nœud ET et correspond au but du problème. Les descendants de ce nœud sont des nœuds OU correspondant à chacun des sous-buts. Les descendants de ces nœuds sont les ensembles des pré-conditions des actions qui établissent ces sous-buts. Un nœud OU est résolu si l'effet qui lui correspond appartient à l'état initial ou si un de ses descendants est résolu. Un nœud ET est résolu si tous ses descendants sont résolus. Un ensemble d'effets utiles pour résoudre un problème est donc constitué par les feuilles d'un sous-arbre solution du graphe. Le but est de trouver les ensembles les plus petits possibles pour l'inclusion, mais comme il s'agit d'un problème NP-complet, on doit faire des approximations. De plus, comme on ne tient pas compte des retraits des actions, prendre un seul de ces ensembles sera a priori trop restrictif pour constituer l'état initial. Plusieurs possibilités sont alors envisagées pour sélectionner un ensemble assez grand, le principe étant de prendre un ensemble suffisamment réduit pour améliorer les performances globales du planificateur, mais suffisamment grand pour ne pas rejeter des effets absolument nécessaires.

– Le planificateur LCGP[9] [Cayrol *et al.* 00] procède de manière similaire à GraphPlan en construisant incrémentalement un graphe avant d'en extraire un plan solution. Le graphe construit par GraphPlan est un sous-graphe de celui construit par LCGP à profondeur égale. La préférence donnée au temps de réponse sur la qualité des plans produits est un point essentiel de LCGP : le temps de production d'un plan solution (pas forcément optimal) est plus réduit pour LCGP quand GraphPlan est incapable de produire une solution au bout d'un temps considérable. GraphPlan n'impose aucune hypothèse sur la façon dont l'exécution peut se dérouler dans le monde réel : on connaît pas la durée d'exécution d'une action, on ne sait pas combien de temps met un effet pour être établi, ni combien de temps une pré-condition doit rester vraie pour permettre d'exécuter une action... Toutes ces incertitudes obligent GraphPlan à contraindre fortement le choix des actions

[9] Least Committed GraphPlan.

afin que le résultat de l'exécution en série d'un ensemble d'actions (dans n'importe quel ordre) conduise au même état que l'exécution en parallèle de ces mêmes actions. LCGP montre qu'on peut modifier la propriété d'indépendance entre les actions simultanées d'un ensemble. Les résultats de comparaison de GraphPlan et LCGP sont du côté du dernier ; la perte d'optimalité est très largement compensée par le gain de temps (LCGP est 1000 fois plus rapide que GraphPlan dans certains problèmes). De plus, l'espace de recherche dans l'algorithme LCGP est plus compact (moins de niveaux que GraphPlan, chacun plus dense) et entraîne dans certains domaines une telle amélioration des performances que la perte de la propriété d'optimalité qui en résulte est négligeable. Les modifications faites dans les algorithmes de GraphPlan ainsi que les tests expérimentaux sont détaillés dans [Vidal *et al.* 99, Cayrol *et al.* 00].

– Une possibilité pour prendre en compte en partie les effets négatifs des actions est proposée pour le planificateur AltAlt [Nigenda *et al.* 00, Nguyen & Kambhampati 00, Nguyen & Kambhampati 01] par le calcul complet du graphe de planification, c'est-à-dire prenant en compte les exclusions mutuelles. Les effets et les actions n'apparaissent que plus tardivement dans le graphe fourni par le problème relaxé. Plusieurs heuristiques sont proposées : la plus performante fait intervenir le niveau d'apparition des effets pour calculer la longueur d'un plan sans prendre en compte les retraits des actions, et fait aussi intervenir le niveau de disparition des exclusions mutuelles entre effets afin d'ajuster l'estimation de la longueur du plan, ce qui est une manière indirecte de prise en compte des retraits des actions.

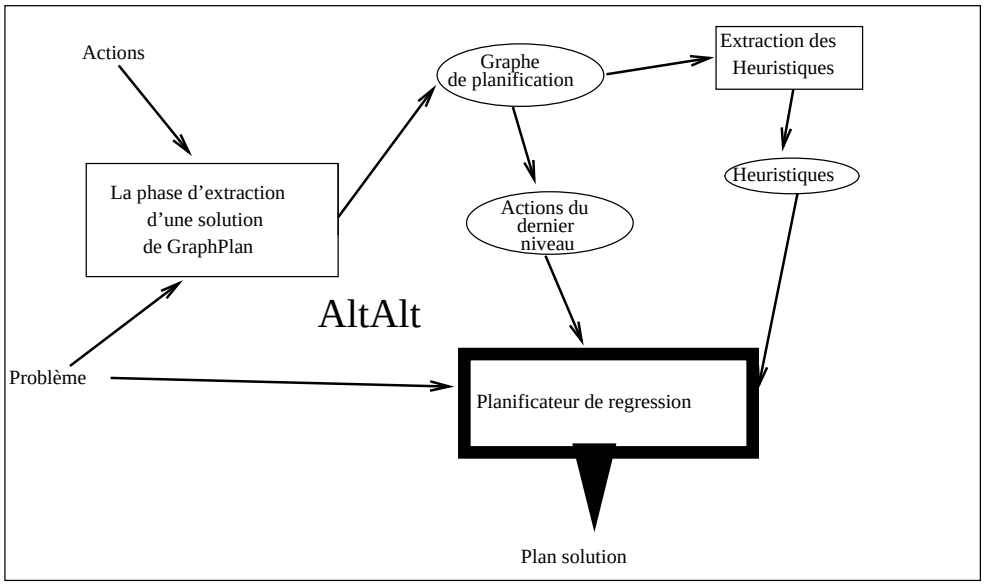

Fig. 2.7 – L'architecture de AltAlt (d'après [Nigenda *et al.* 00])

Le planificateur AltAlt, dont l'architecture est illustrée dans la figure 2.7, est basé sur une combinaison de GraphPlan et de techniques de recherche dans les espaces d'états avec chaînage arrière. Le problème et les actions sont d'abord introduits dans le système GraphPlan pour construire un graphe de planification en temps polynomial. Ce graphe est ensuite introduit dans un système d'extraction des heuristiques afin de définir les heuristiques effectives et admissibles, basées sur des théories développées dans [Nguyen & Kambhampati 00]. Enfin, le problème, les actions du dernier niveau du graphe de planification produit par GraphPlan et les heuristiques choisies sont introduits dans le planificateur afin de produire le plan-solution.

2.6.4 UCPOP

Le planificateur UCPOP[10] [Penberthy & Weld 92, Weld 94] utilise, comme langage de description des domaines et des problèmes, un sous-ensemble important du langage ADL [Pednault 89]. Le planificateur ainsi obtenu, basé sur SNLP[11] [McAllester & Rosenblitt 91], est sain [12] et complet[13]. L'algorithme UCPOP part d'un plan qui contient l'état initial et l'état final et essaye de le compléter en additionnant des actions et des contraintes jusqu'à ce que toutes les pré-conditions soient satisfaites. La boucle principale de cet algorithme est composée de deux choix :

1. Si l'algorithme ne satisfait pas une pré-condition, alors tous les effets qui sont peut-être conditionnés par cette pré-condition sont considérés comme vrais. Ensuite, UCPOP choisit un effet non-déterministe et ajoute un lien causal[14] [McAllester & Rosenblitt 91] au plan pour valider ce choix.
2. Si une autre étape devient perturbée par la pré-condition supportée par le lien causal, alors UCPOP choisit une méthode pour résoudre ce problème : ré-ordonner les étapes du plan, ajouter des sous-buts ou bien ajouter des contraintes d'égalité.

Quand UCPOP crée un lien causal pour tous les buts dans le plan, il retourne une solution. Un des principaux inconvénients de ce planificateur provient de sa stratégie de résolution de conflits ; après chaque raffinement d'un plan partiel, il faut choisir le prochain raffinement à effectuer. En effet, il entrelace les deux stratégies de raffinement suivantes :

[10] Universal quantification and Conditional effects to a Partial Order Planner, prononcé yoo-see-pop.

[11] Systematic NonLinear Planner.

[12] Un planificateur est sain ssi tous les plans qu'il peut produire sont corrects c.-à-d. que l'exécution de ces plans permet, partant de l'état initial du monde, d'aboutir à un état but.

[13] Un planificateur est complet ssi il produit un plan permettant de résoudre le problème lorsqu'un tel plan existe.

[14] Un lien causal est une liste $< S_i, e, r, S_j >$, notée $S_i \xrightarrow{e,r} S_j$, où r est une pré-condition de S_j, e est un effet de S_i et $\exists q \in \theta_e$ tel que q est fusionné avec r.

1. Raffinements dans les espaces de plans : après avoir ajouté une étape ou une contrainte de précédence dans le plan partiel, il vérifie qu'aucune contrainte n'est violée. Plus il y a de contraintes dans le plan, plus cette opération est coûteuse ; et plus il y a de conflits, plus le nombre de plans partiels générés pour résoudre ces conflits est important.
2. Raffinements auxiliaires de pré-satisfaction : il résout un conflit provoqué par une étape sur une contrainte auxiliaire. Comme précédemment, plus il y a de conflits et plus la résolution est coûteuse.

Le problème essentiel pour ce choix provient des conflits qui ne sont pas encore résolus. Si l'on ne s'en aperçoit pas immédiatement, tous les choix de raffinement que l'on peut faire conduisent inévitablement à un échec. Par exemple sur certains problèmes, 98% des nœuds examinés dans l'arbre de recherche par UCPOP correspondent à des plans partiels comportant un conflit impossible à résoudre [Joslin & Pollack 96]. De même, il se peut qu'un conflit ne puisse être résolu que d'une seule manière (par exemple, uniquement par promotion par rapport à une étape). Si l'on effectue ce choix immédiatement, il n'y aura pas d'autre possibilité à examiner en cas de retour arrière. Par contre, si on effectue un autre raffinement produisant plusieurs plans partiels, il faudra répéter le même choix de résolution de ce conflit pour tous ces plans partiels et tous leurs descendants dans l'arbre de recherche.

2.6.5 Autres planificateurs classiques

Plusieurs autres planificateurs classiques, basés en général sur STRIPS ou sur GraphPlan, ont montré leur efficacité de résoudre un certain nombre de problèmes. Parmi les planificateurs les plus connus dans ce type de planification, nous citons :

- ProPlan[15] [Fourman 00], inspiré de GraphPlan, utilise la recherche dans les espaces d'états mais au lieu d'explorer tous les états individuels, il se concentre sur les ensembles d'états accessibles. Dans ProPlan, les actions sont propositionnelles et sont exécutées séquentiellement ; une seule action par étape du plan. L'algorithme de ProPlan prend en entrée une situation et un but et fournit en sortie des plans qui satisfont le but. Il est divisé en deux étapes principales :
  1. La construction d'une structure de données composée de l'espace des états. Cette structure joue le rôle du graphe de planification de GraphPlan.
  2. La recherche des solutions se fait en définissant une fonction récursive qui cherche les plans dont le nombre d'états est minimal.

  La performance de ProPlan dépend linéairement de plusieurs facteurs : la taille de la base de connaissance qui représente la situation (pré-conditions, effets, buts), le nombre d'ac-

[15] Propositional Planning.

tions et le nombre des états dans le plan minimal.

– PL-PLAN [P. Fournier-Viger and L. Lebel] est un planificateur réalisé en 2004 par Philippe Fournier-Viger et Ludovic Lebel. Ils ont expérimenté plusieurs techniques de planification classique de recherche dans un espace d'états. Le planificateur réalisé comprend huit algorithmes : une implantation de l'algorithme GraphPlan, une recherche en chaînage avant naïve et six algorithmes basés sur la théorie des ordres partiels. À noter que le code de PL-PLAN n'est pas optimisé et que les quatre algorithmes basés sur les ordres partiels qui incorporent la technique des *persistant sets* ne trouvent pas toujours la solution optimale.

2.7 Méthodes de recherche

Planifier, c'est avant tout faire l'exploration d'un espace afin d'y trouver un plan permettant de passer de l'état initial à l'état final souhaité. Les choix qui sont faits lors du développement de l'algorithme de recherche influenceront à la fois la qualité des plans produits et le temps nécessaire à leur production. Un bon plan est un plan *correct*, c'est-à-dire que toutes les préconditions de tous les opérateurs du plan sont vérifiées, et *cohérent*, c'est-à-dire qu'il n'y a aucune contradiction dans l'ordonnancement des opérateurs. Le temps qu'un algorithme passe à explorer une bonne solution constitue toujours un bon investissement puisqu'il permettra d'obtenir un bon résultat. Par conséquence, éviter d'explorer de mauvaises solutions permet de réduire le temps de calcul. La taille de l'espace de recherche et la manière dont l'exploration est faite sont donc cruciales. Les deux stratégies de recherche les plus connues sont la recherche dans l'espace des états [Farreny 95] et la recherche dans l'espace des plans [Weld 94, Kambhampati 97]. L'exploration du graphe peut être réalisée par les techniques connues de parcours dans les graphes [Weld 94] : les recherches avant, arrière, largeur d'abord,...

La recherche par l'avant part de l'état initial et considère tous les états pouvant être atteints à partir de là. La recherche continue ainsi jusqu'à l'état final. L'ordre de grandeur de la taille d'un espace de recherche pour un plan de n étapes et de m opérateurs applicables pour chacune des étapes est égal à $\mathcal{O}(m^n)$. Afin de réduire la taille de l'espace de recherche, nous pouvons faire une recherche arrière dans le but d'éliminer tous les opérateurs qui n'ont rien à voir avec l'objectif à atteindre. La solution adoptée par les planificateurs modernes consiste à faire une recherche arrière dans l'espace des plans plutôt que dans l'espace des états.

Pour déterminer la suite d'actions qui permet de passer de l'état initial à l'état but cherché, nous présentons dans les deux sections suivantes deux manières d'aborder la recherche d'une solution.

2.7.1 Recherche dans les espaces d'états

On appelle état du monde une situation du monde à un moment donné. Il est constitué d'un ensemble de valeurs de vérité de faits qui sont représentés par une proposition logique. On représente les différents changements dans un graphe. Dans la planification par recherche dans les espaces d'états [Farreny 95], les nœuds du graphe de recherche représentent les états successifs du monde du problème de planification et les arcs, les actions qui permettent de passer d'un état à un autre. L'algorithme tente de construire un chemin qui permettre de passer de l'état initial à l'état but. Ce chemin correspond à un sous-graphe du graphe complet de toutes les actions possibles sur tous les états. Il est à noter que certains états du monde ne seront pas accessibles à partir d'un état initial donné. La recherche se termine quand l'état courant contient les buts à atteindre. En général, les planificateurs utilisant cette approche n'offrent pas de garantie sur l'optimalité du nombre d'actions des plans-solutions.

2.7.2 Recherche dans les espaces de plans

Dans la planification par recherche dans les espaces de plans partiels [Weld 94, Kambhampati 97], les nœuds du graphe de recherche représentent des plans partiels et les arcs, les opérations d'extension de ces plans. L'algorithme cherche à étendre un plan initial qui présente le problème posé pour obtenir un plan-solution. Il se termine lorsque toutes les pré-conditions de toutes les actions présentes dans le plan courant sont produites par d'autres actions du plan et que ce dernier ne comporte plus aucune action qui puisse potentiellement interférer avec une autre. Si des interactions entre sous-buts subsistent, une relation d'ordre est établie, ou des variables peuvent être instanciées. Il peut également être nécessaire de faire un retour arrière. La stratégie utilisée pour l'extension des sous-buts est celle de l'engagement minimum ("least commitment") [Weld 94] : on suppose que le problème est bien décomposable et que les interactions entre sous buts seront gérées lorsqu'elles apparaîtront au fur et à mesure du développement des plans partiels. Cette approche demeure ainsi intéressante, car elle permet d'obtenir des plans possédant un haut degré de flexibilité.

2.7.3 Complexité

Comme nous l'avons vu précédemment, la planification est essentiellement un problème de recherche dans un espace (d'états du monde, de sous-problèmes ou de plans) : le processus passe en revue l'espace de recherche pour produire un plan qui puisse s'appliquer à l'état initial et le transformer de manière à satisfaire le but. Si l'utilisation d'heuristiques permet de restreindre cet espace de recherche, la complexité du problème est cependant importante et dépend des méthodes employées. En définissant la planification comme un problème de recherche dans un graphe d'états, Korf [Korf 87] donne les complexités des différentes stratégies de résolution que

l'on peut employer :

- La recherche en largeur d'abord fournit une solution en un temps $\mathcal{O}(bp)$ mais utilise aussi la mémoire en $\mathcal{O}(bp)$ (b est le facteur de branchement et p la longueur du plan-solution comportant le nombre minimal d'opérateurs).
- La recherche en profondeur d'abord utilise linéairement l'espace mais nécessite une profondeur arbitraire pour s'arrêter ; si la valeur de cette dernière est plus petite que p on ne trouve pas de solution, si elle est plus grande on travaille plus que nécessaire et le plan trouvé n'est pas forcément minimal.
- La recherche en profondeur itérative combine les avantages des deux algorithmes précédents en effectuant une série de recherches en profondeur d'abord avec $p = 1$ puis $p = 2$,... Cet algorithme fournit un plan solution minimal en un temps $\mathcal{O}(bp)$ et en utilisant $\mathcal{O}(p)$ espace.
- En utilisant des fonctions heuristiques, la complexité en temps de ces algorithmes peut passer de $\mathcal{O}(bp)$ à $\mathcal{O}(hp)$ (h étant le facteur de branchement heuristique, avec $h < b$).

2.8 Planification et CSP

Certaines classes de problèmes de planification peuvent être modélisées comme un problème de satisfaction de contraintes (CSP[16]) [Bessière 92, Russell & Norvig 03, Dvorak *et al.* 04]. Un tel problème est défini par la donnée de n variables discrètes $x_1, ..., x_n$ décrivant des domaines finis $D_1, ...D_n$, et de m contraintes $c_1, ..., c_m$ qui spécifient explicitement ou implicitement les tuples consistants $c_k(x_i, ..., x_j) \subseteq D_{i_1} \times ... \times D_{i_k}$. Un CSP est *binaire* si toutes ses contraintes ne portent que sur deux variables (on les notera : $c_{ij}(x_i, x_j)$). Il est alors représentable par un graphe dont les nœuds sont étiquetés par les variables et les arcs les contraintes ; noter que c_{ji} est la contrainte symétrique de c_{ij}.

Une solution à un problème de satisfaction de contraintes est une assignation de valeurs aux variables qui satisfait toutes les contraintes. Si une telle solution existe, alors le CSP est consistant. Le test de la consistance d'un CSP est un problème NP-complet, même pour un CSP binaire. Un CSP est $k-$consistant si pour chaque tuple consistant de $(k-1)$ variables, il est possible de trouver une assignation, consistante avec ce tuple, à la k^{e} variable. Ainsi la 2-consistance signifie que pour toute valeur de x_i dans son domaine, on peut trouver une valeur de x_j telle que le couple (x_i, x_j) soit consistant. Vérifier la 2-consistance revient à réduire les domaines D_i pour en éliminer toute valeur qui n'est consistante avec aucune valeur d'un autre D_j. Cette opération, dite de filtrage local par 2-consistance, donne une condition nécessaire mais non suffisante de consistance globale : si après filtrage un des D_i est vide, alors le CSP est inconsistant. Cependant, même si aucun domaine filtré n'est vide, la consistance globale n'est pas assurée. Le filtrage d'un CSP pour la k-consistance se fait en $\mathcal{O}(nk)$. Un CSP $n-$consistant

[16] Constraint Satisfaction Problem.

est consistant ; on peut en trouver une solution en temps linéaire. Le filtrage d'un CSP par k-consistance, pour k petit (2 ou 3), est souvent utilisé pour faire des vérification à faible coût, en particulier lors d'ajout incrémental de contraintes.

2.9 Planification hiérarchique

Plusieurs planificateurs développés dans le cadre d'applications réelles utilisent les réseaux de hiérarchisation de tâches (HTN[17]) [Erol *et al.* 94, Hayashi *et al.* 04, Besse-Patin 05]. La principale différence avec la planification classique et la planification hiérarchique est que cette dernière essaie de décomposer des tâches de haut niveau en tâches de plus bas niveau là où la planification classique cherche juste à assembler des actions pour atteindre des buts. De plus, un but est plutôt spécifié, dans la planification hiérarchique, comme une tâche de haut niveau, que comme un ensemble de littéraux à obtenir.

```
HTN_Plan(Plan)
début
    si Plan contient des conflits alors
        si il n'existe pas de manière de résoudre les conflits alors
            Échec
        sinon
            Choisir une manière de résoudre les conflits et l'appliquer (Choisir indique un
            point de Backtrack)
        fin
    fin
    si Plan ne contient que des tâches élémentaires alors
        Retourner (Plan)
    fin
    Sélectionner une tâche non-élémentaire t ∈ Plan
    Choisir une décomposition E de t
    NouveauPlan ← Remplacer t par E dans Plan
    HTN_Plan(NouveauPlan)
fin
```

Algorithme 5 – L'algorithme de la planification hiérarchique (HTN)

L'algorithme de planification (représenté dans l'algorithme 5) consiste donc à décomposer chaque tâche en tâches élémentaires, tout en vérifiant qu'elles n'aient pas d'interactions négatives entre elles. La planification se termine lorsque le réseau ne contient que des tâches élémentaires et

[17] Hierarchical Task Network.

lorsque l'ensemble des contraintes d'ordre relatives à la gestion des conflits est consistant (c.-à-d. qu'il n'existe pas de boucles).

Le temps et la gestion de données métriques ne posent pas de réelles difficultés pour les planificateurs HTN. Ces contraintes peuvent être directement spécifiée à l'intérieur des tâches ou alors évaluées pas le biais de test de consistance de l'algorithme. Seulement, selon [Smith *et al.* 00], les planificateurs HTN essuient pour le moment trois critiques importantes :

1. **Sémantique :** il n'existe pas de sémantique bien définie pour la décomposition de tâche, cela a pour conséquence de rendre difficile tout jugement de la complétude ou de la consistance d'un plan.
2. **Conception :** la conception de ce type de planificateur nécessite de prévoir et d'analyser toutes les tâches possiblement existantes et décomposables. Il est vraiment difficile d'une part, de produire la liste exhaustive de toutes les décompositions d'une tâche, et d'autre part, à chaque ajout de fonctionnalité au système, il faudra prévoir et lister toutes les nouvelles décompositions à ajouter pour utiliser au mieux les fonctionnalités du système.
3. **Fragilité :** les planificateurs HTN sont incapables de prendre en compte des tâches non-explicitement prévues par le concepteur, même si les tâches élémentaires sont suffisantes pour construire un plan correct.

2.10 Planification et exécution

Le contrôle d'exécution de plans est un problème particulièrement difficile lorsqu'il doit être effectué à bord de systèmes autonomes tels que des robots. Un tel système doit disposer de processus de délibération pour engendrer des plans qui réalisent les buts de la mission tout en respectant des délais et des contraintes de précédence. La figure 2.8 représente un exemple d'un système de planification qui intègre un contrôleur d'exécution.

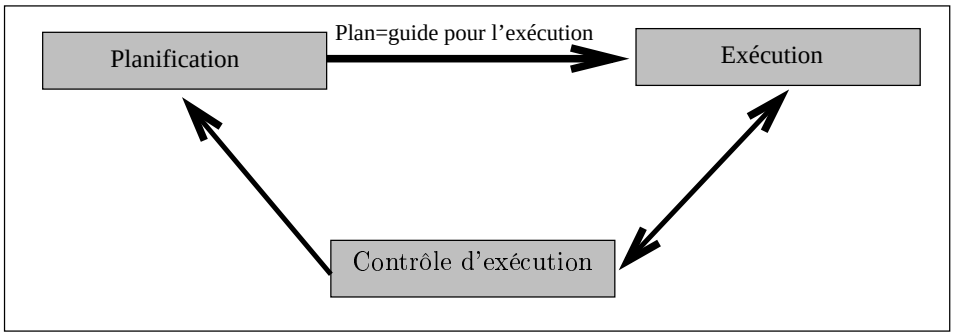

Fig. 2.8 – Planification/exécution

Dans tout ce qui précède, nous n'avons que très peu parlé des méthodes qui ont été employées

pour remédier aux échecs lors de l'exécution du plan (dûs à des événements imprévus, au manque de précision dans la description des opérateurs ou du monde,...). Nous ne traitons pas ce problème dans cette thèse mais, nous citons brièvement les principales techniques utilisées en cas d'échec :

1. Re-planifier lorsqu'une différence entre l'état du monde attendu et l'état constaté apparaît.
2. Lier différemment les processus de génération et d'exécution des plans. Ceci rend le planificateur plus sensible aux interactions, mais plus réactif aux changements imprévus de l'univers [Mouaddib & Zilberstein 97, Lemai & Ingrand 04].
3. Anticiper les échecs et planifier pour les résoudre avant qu'ils ne se présentent [Bresina & Washington 00].
4. Lier la planification, l'exécution d'actions et la surveillance de l'exécution dans un même système.
5. Contrôler la situation en lançant un système d'exécution indépendant dès qu'il est possible d'agir sans l'aide d'un plan. Ce dernier sert simplement à augmenter les capacités du système à atteindre les buts.

Certaines erreurs lors de l'exécution paraissent inévitables : elles sont généralement dues à des événements imprévus du monde réel sur lesquels on ne peut avoir aucun contrôle (figure 2.9).

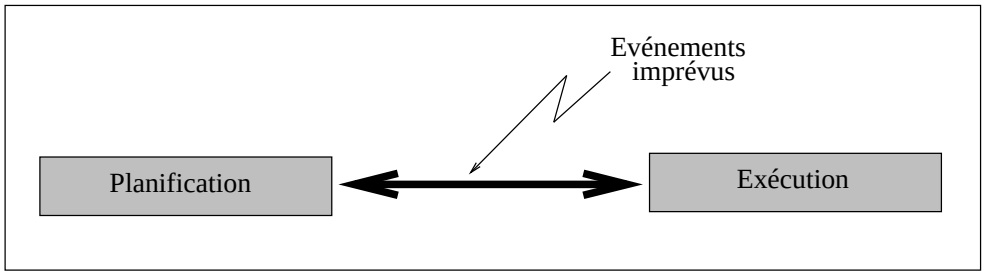

Fig. 2.9 – Événements imprévus entre la planification d'une action et son exécution

Des techniques portent sur le moyen d'en limiter le nombre en entrelaçant la planification et l'exécution de façon à réduire le délai entre la planification d'une action et son exécution ainsi que sur les moyens à mettre en œuvre pour réagir de manière adéquate à ces événements et atteindre le but initialement fixé. En préambule à ces travaux, cette section présente une synthèse des principaux systèmes de planification et d'exécution qui existent. Nous expliquons leurs processus de planification et de suivi d'exécution ainsi que le délai qui sépare la planification d'une action de son exécution. Ce critère a été prédominant pour caractériser la plus ou moins grande réactivité de ces systèmes, c'est-à-dire leur capacité à s'adapter aux évolutions imprévues de l'univers. Cette caractéristique nous conduit à définir trois catégories :

– Les systèmes traditionnels qui ne débutent l'exécution que lorsque le plan est entièrement construit ; ce sont ceux pour lesquels le délai entre planification et exécution est maximal.

– Ceux qui entrelacent planification et exécution, c'est-à-dire qui planifient plusieurs actions puis les exécutent tout en poursuivant la planification.
– Ceux qui réduisent ce délai en exécutant des plans pré-compilés en réaction à une situation précise.

Nous présentons dans la suite quelques planificateurs intéressés par la planification et l'exécution.

2.10.1 SIPE

SIPE[18] [Wilkins 88] est l'un des premiers systèmes ayant une architecture à deux niveaux : il intègre un planificateur et un système réactif. Ce dernier se compose de contrôleurs capables de générer un comportement réactif en fonction des informations fournies par les capteurs et effecteurs du robot mobile associé. La figure 2.10 illustre l'architecture à deux niveaux de SIPE.

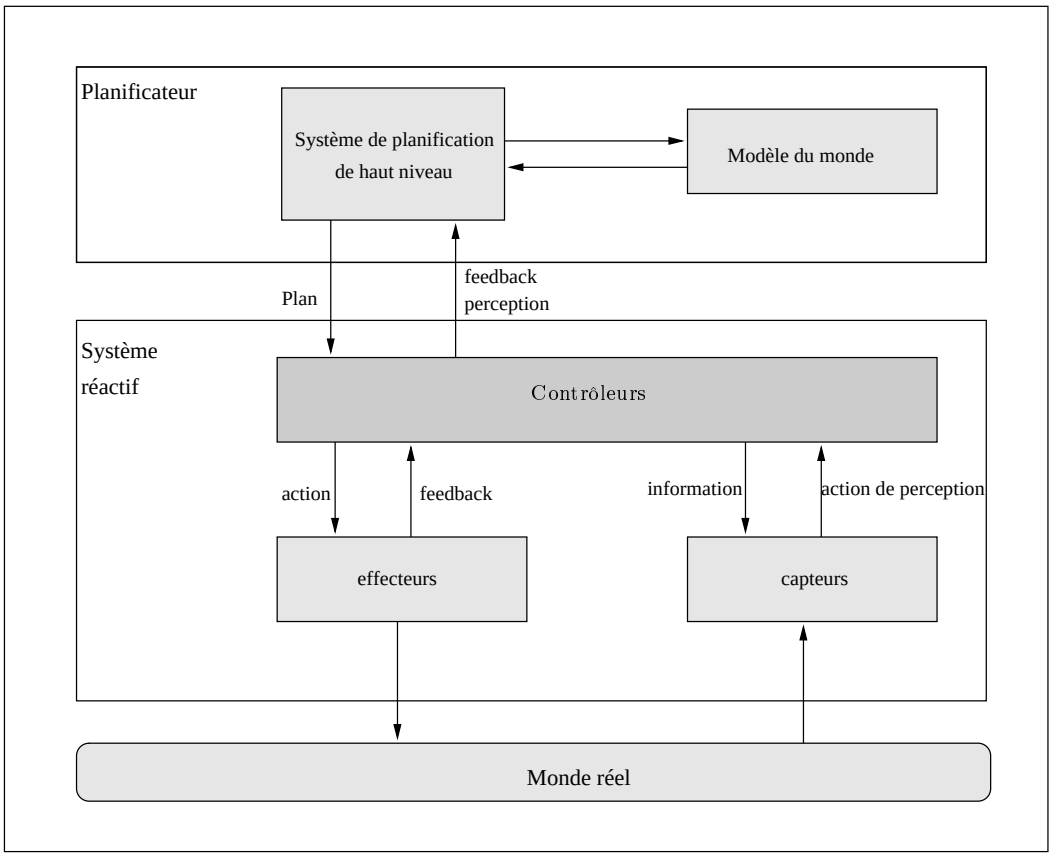

Fig. 2.10 – L'architecture à deux niveaux de SIPE (d'après [Wilkins 88])

SIPE réalise un compromis entre une représentation par frame et la logique des prédicats. Une hiérarchie de nœuds (similaire aux réseaux sémantiques) permet en effet de représenter les

[18] System for Interactive Planning and Execution.

propriétés invariantes des objets. Chaque nœud peut avoir des attributs propres et hériter des propriétés d'autres nœuds de la hiérarchie. Les valeurs des attributs peuvent être des nombres, des pointeurs vers d'autres nœuds, des mots-clés pour le système, des chaînes de caractères,... Une restriction de la logique des prédicats permet la représentation des propriétés des objets qui varient lorsque des actions sont effectuées et sert également à décrire les pré-conditions et les effets des opérateurs ainsi que les buts à atteindre. La représentation séparée des propriétés invariantes du domaine permet de réduire le nombre d'expressions mises en jeu dans le système et autorise ainsi une plus grande efficacité.

SIPE fait l'hypothèse de STRIPS (toutes les relations non explicitement mentionnées dans une liste d'ajouts/retraits ne sont pas modifiées) et celle du monde clos (tout prédicat non présent dans la base de fait est faux). SIPE repousse les limites du postulat de STRIPS grâce à l'emploi d'opérateurs de déduction (domain rules) qui permettent de déduire de nouveaux faits de l'état courant de l'univers. Ceci permet de ne pas avoir à représenter explicitement tous les effets des opérateurs (en particulier les effets de bord qui deviennent extrêmement nombreux lorsque la complexité du domaine s'accroît) mais rend le planificateur incorrect dans le cas général [Chapman 87]. Les contraintes permettent au planificateur de manipuler (avant instanciation) des variables satisfaisant certaines propriétés (exprimées sous forme de contraintes dans un langage spécifique). L'usage des contraintes permet d'obtenir des opérateurs et des plans plus faciles à comprendre et à rédiger ainsi que d'effectuer des instanciations de variables en réduisant le nombre de backtracks. En contrepartie, on augmente la complexité du système (gestion, combinaison et diffusion des contraintes dans le procedural net). SIPE permet :

- L'utilisation d'un langage général de description des contraintes permettant la modélisation d'un grand nombre de domaines.
- La possibilité d'évaluer les contraintes sur les variables avant leur instanciation, l'ensemble ainsi fourni pouvant être utilisé pour la planification avant que ses éléments ne soient complètement identifiés.
- La possibilité de faire varier les descriptions partielles avec le contexte : ceci permet l'exploration simultanée de branches alternatives du plan.
- D'exprimer les ressources nécessaires à l'exécution d'une action et de raisonner dessus. Ces ressources sont affectées à l'action pendant toute la durée de sa réalisation et libérées ensuite. Ceci décharge le concepteur de la base de connaissances de la tâche difficile qu'était leur formalisation au travers des pré-conditions des opérateurs. Ceci évite également au système des critiques spécifiques aux conflits entre ressources : dans SIPE, les conflits qui leur sont dus sont détectés avant même que le plan ne soit détaillé, la TOME (Table Of Multiple Effects) n'est dressée qu'en cas d'interactions supplémentaires, pour tenter d'y remédier.

Dans SIPE, quand l'exécution d'une branche permet de satisfaire un but d'une branche parallèle,

le planificateur peut en tirer parti en exécutant d'abord les actions de la première branche, minimisant ainsi le nombre d'actions du plan. De plus, l'introduction de ressources dans les opérateurs (et même de ressources partageables sous certaines conditions exprimées par des formules logiques) permet de limiter le nombre d'interactions qui apparaissent dans le plan.

2.10.2 SPEEDY

SPEEDY[19] [BASTIÉ 97] est essentiellement basé sur l'entrelacement des processus de planification et d'exécution et nécessite l'intégration, au sein du même système, de trois processus principaux (figure 2.11) ; ces processus sont tous incrémentaux et fonctionnent de manière concurrente :

- Planification linéaire (algorithme général de planification permettant au système de produire les actions qu'il exécute afin d'atteindre le but).
- Recherche des actions exécutables en parallèle (processus qui permet d'optimiser le temps d'exécution du plan et de mettre à profit les capacités de parallélisme du système robotique utilisé).
- Suivi d'exécution (processus permettant de lancer l'exécution des actions exécutables, de détecter les événements anormaux pouvant survenir pendant l'exécution, et d'y réagir de manière appropriée).

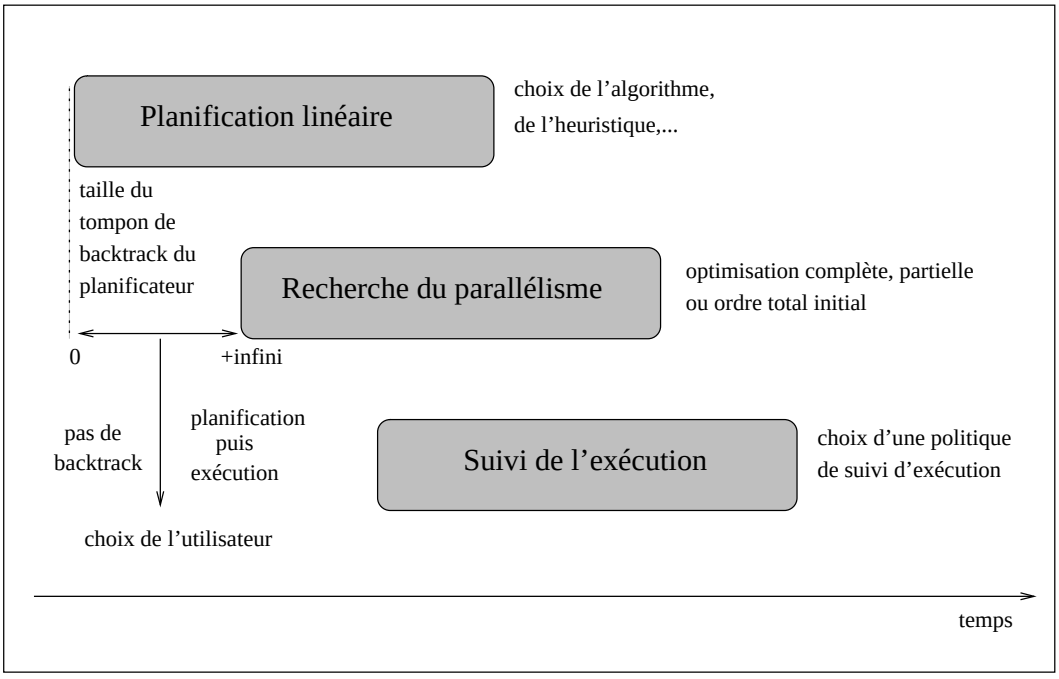

FIG. 2.11 – La configurabilité du système SPEEDY (d'après [BASTIÉ 97])

[19] Système de Planification et d'Exécution en Environnement DYnamique.

Le processus de planification produit (action par action) une séquence d'actions dont l'exécution doit permettre, à partir de l'état initial du monde de conduire celui-ci dans un état satisfaisant l'ensemble des buts donnés par l'utilisateur lors de l'initialisation du système. Il fait appel à un processus de planification linéaire. Dans le cas d'un entrelacement de la planification et de l'exécution, ce type de processus facilite notamment la détermination des actions exécutables. Les actions sont mémorisées, au fur et à mesure de leur production, dans une table triangulaire[20], laquelle permettra d'isoler les contraintes d'ordre nécessaires entre les actions du plan et donc de déterminer quelles actions peuvent être exécutées en parallèle. Par contraste, le processus de suivi d'exécution développé dans SPEEDY est général (il n'utilise aucun programme ou règle pré-compilés) et autonome (il ne fait pas appel à l'utilisateur). Il est capable de détecter tous les types d'anomalies et peut, contrairement aux systèmes existants, les différencier afin d'y réagir de manière appropriée : les échecs sont classés en fonction de leur importance pour l'obtention du but. SPEEDY propose également plusieurs stratégies différentes en réaction aux échecs ou aux imprévus. Le suivi d'exécution peut y être configuré par l'utilisateur : ce dernier peut définir sa propre politique de suivi d'exécution et choisir les stratégies à appliquer différents types d'anomalies, ainsi que leur ordre. SPEEDY est de plus capable de traiter des problèmes d'effecteurs, notamment en remplaçant les effecteurs défectueux par d'autres, remplissant les même fonctions, mais en bon état.

2.10.3 SimPlanner

SimPlanner [Sapena & Onaindia 02], basé sur STRIPS, est un planificateur qui intègre les deux étapes planification et exécution pour avoir une planification en ligne. La technique de la planification en ligne permet de commencer l'exécution d'un plan avant qu'il soit complet. Dès que la première action est déterminée, l'exécution peut commencer et la planification continue en parallèle avec l'exécution pour déterminer les nouvelles actions à exécuter. La figure 2.12 représente le planificateur SimPlanner. L'algorithme, divisé en quatre étapes, commence par l'état initial en choisissant l'action à exécuter :

1. Sélection des buts non achevés : ce sont les buts qui ne sont pas dans l'état *vrai* à l'état courant. C'est-à-dire qu'ils appartiennent à l'état final, mais ne sont pas encore atteints.

[20] Une table triangulaire (TT) est la moitié inférieure gauche d'un tableau $n \times n$ qui représente un plan linéaire avec les pré-conditions et les effets de chacune de ses actions. Le plan se trouve sur la diagonale. La case C_{ij} (colonne i, ligne j) contient les pré-conditions de l'opérateur op_j qui sont les faits ajoutés par op_i. La première colonne (colonne 0) contient les conditions nécessaires à l'exécution du plan qui existent dans l'état initial et la dernière ligne, les effets finaux du plan. La TT permet de rechercher la valeur de vérité d'un fait, d'obtenir l'état courant du monde et de déterminer l'état de l'univers avant l'ajout de l'action A_i. Elle peut être utilisée pour le contrôle d'exécution, mais aussi pour la généralisation de plan, la construction de macro-opérateurs pour l'explication de plan ou encore pour la recherche du parallélisme dans un plan d'actions.

2. Détermination des plans approximatifs : un plan approximatif est déterminé séparément pour chaque but non achevé. En effet, le problème de planification $P = (O, I, G)$, où O est l'ensemble d'actions, I est l'état initial et G est l'ensemble des buts, est composé de m sous-problèmes $P_1 = (O, I, g_1)$, $P_2 = (O, I, g_2)$,..., $P_m = (O, I, g_m)$.
3. Regroupement des plans approximatifs : les plans approximatifs sont regroupés selon leurs actions initiales dans des groupes appelés branches tels que tous les plans approximatifs d'une même branche partagent les mêmes actions initiales.
4. Sélection de l'action à exécuter : les branches sont ordonnées selon certains critères[21]. La prochaine action à exécuter sera la première action de la branche ordonnée en premier.

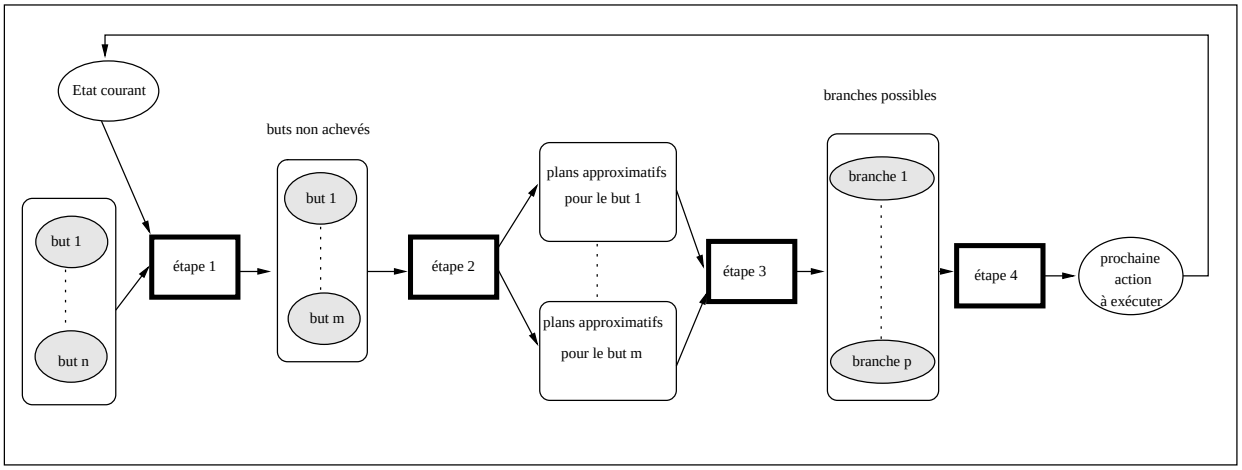

Fig. 2.12 – Le planificateur SimPlanner (d'après [Sapena & Onaindia 02])

Dans [Sapena & Onaindia 02], les auteurs détaillent ces étapes de planification et d'exécution. Ils ont montré que SimPlanner choisit les actions à exécuter en temps raisonnable, mais pas toujours de bonnes qualités. Puisque l'exécution prend en général plus de temps que la planification elle même, il est possible d'optimiser le choix des actions restantes dans le temps de l'exécution des actions déjà choisies. Cette hypothèse est correcte à partir de la deuxième action.

2.11 Limites de la planification classique

La planification dite classique s'intéresse à la génération de plans dans un univers statique et totalement connu à l'aide d'actions déterministes, sans gestion de ressources (temporelles ou autres). On se situe dans un cadre très simplifié par rapport aux contraintes du monde réel, et les principaux travaux dans ce domaine concernent l'étude et l'amélioration des performances des algorithmes.

[21] Les critères de sélection dépendent du problème à résoudre.

Malgré l'utilisation d'une recherche arrière dans l'espace des plans, la complexité algorithmique des planificateurs classiques reste trop élevée pour qu'ils soient appliqués à des problèmes réels. Elle est au pire des cas NP-Complet ; un algorithme déterministe demande un temps exponentiel pour trouver une solution ou pour décider qu'il n'y en a pas de valide. Russel et Norvig [Russell & Norvig 03] donnent l'exemple de la construction d'une maison comme problème trop complexe pour une planification classique. La plupart de planificateurs classiques supposent que le monde est totalement observable et certain, et ils ne sont pas capables de :

1. traiter des actions avec des durées,
2. permettre aux actions de s'exécuter simultanément,
3. ajouter des effets probabilistes.

De plus, ils limitent l'évaluation des plans à une fonction binaire basée sur le succès ou l'échec d'atteinte de l'objectif. Or, certaines situations nécessitent une évaluation plus large comme par exemple le coût et le temps d'exécution, les ressources consommées, la probabilité de la réussite de l'exécution, etc.

Dans le chapitre suivant, nous traitons les planificateurs qui permettent de prendre en compte un de ces critères d'évaluation : il s'agit du temps.

Chapitre 3

La planification temporelle

Dans les planificateurs qui utilisent une représentation de type STRIPS, les actions sont le plus souvent considérées comme étant discrètes : leurs pré-conditions sont vraies jusqu'au début de leur exécution et leurs effets deviennent vrais dès que l'exécution se termine (les transitions sont supposées instantanées). Cette représentation est très souvent insuffisante pour traiter des problèmes qui nécessitent la prise en compte de contraintes temporelles de durée et d'ordonnancement. Comme c'est le cas de la plupart des problèmes réels de planification, plusieurs systèmes spécifiques ont été développés afin de traiter cet aspect.

À partir du début des années 80, le développement des logiques temporelles (en particulier les logiques de McDermott [McDermott 82] et de Allen [Allen 81, Allen 83]) a permis le développement de planificateurs prenant en compte le temps d'une manière spécifique.

La prise en compte du temps enrichit la planification en termes d'expressivité, mais la complexifie en multipliant les possibilités de conflits.

3.1 DEVISER

DEVISER [Vere 81, Vere 85] est le premier planificateur qui a vraiment tenu compte du temps d'une manière assez détaillée. Il traite les actes, les événements et les inférences en leur adjoignant des spécifications temporelles (durée et fenêtre d'activation). DEVISER est un planificateur à caractère général ; il effectue une planification non-linéaire. Le plan est un réseau procédural hiérarchique : les actes, événements et inférences y sont modélisés suivant le postulat de STRIPS (listes d'ajouts et de retraits). Même si les transitions sont instantanées, des contraintes temporelles de durée (définies par avance ou calculables dynamiquement) et des fenêtres d'activation (laps de temps pendant lequel on peut exécuter l'action) leur sont associées pour permettre la prise en compte du temps.

La syntaxe employée dans DEVISER pour les activités (actions, événements,...) est la suivante :

```
(<Nom_de_l'activité><Type_de_l'activité>
(CONTEXT <Liste_de_littéraux_du_contexte>)
(<Liste_des_préconditions> -> <Liste_des_effets>)
(DURATION <Définition_de_la_durée>)
(WINDOW <Définition_de_la_fenêtre>))
```

Une fenêtre d'activation définit les frontières temporelles maximales et minimales pour la réalisation d'une activité. Elle est la transcription des méthodes des réseaux PERT : ces frontières sont des dates soit de début, soit de fin, au plus tôt ou au plus tard d'une activité. Une fenêtre peut aussi contenir une date idéale optionnelle indiquant un instant de réalisation préféré. La figure 3.1 représente une fenêtre d'activation et une durée dans le planificateur DEVISER.

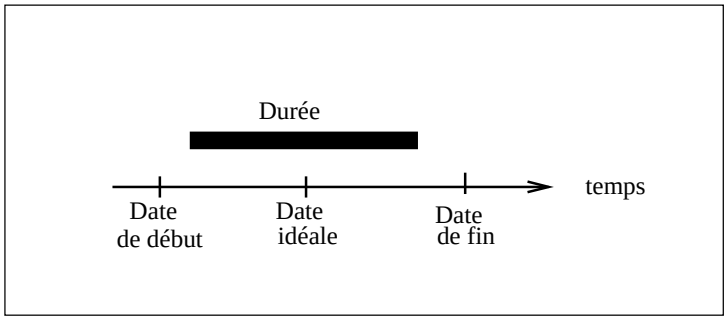

FIG. 3.1 – Une fenêtre d'activation et une durée dans DEVISER

Une fenêtre est un triplet du type :

```
(<Date_de_début_au_plus_tôt><Date_idéale><Date_de_début_au_plus_tard>)
```

où : les dates sont des réels positifs, <Date_idéale> est optionnelle et <Date_de_début_au_plus_tôt> ≤ <Date_idéale> ≤ <Date_de_début_au_plus_tard>.

DEVISER peut également prendre en compte des événements qui se produisent à une heure déterminée (scheduled events), par exemple l'heure d'arrivée d'un avion, l'heure du coucher de soleil,... Ils sont introduits dans le réseau avant le début de la planification, les contraintes qu'ils engendrent sont ensuite prises en compte durant tout le processus de génération de plan de manière à résoudre les interactions ou satisfaire les buts.

3.2 IxTeT

Le planificateur IxTeT [LARUELLE 94], développé au laboratoire LAAS à Toulouse en 1994, se place dans le cadre des approches par opérateurs. Les hypothèses de départ sont celles d'un monde prévisible et d'un ensemble d'informations complet, afin de limiter la combinatoire du problème.

C'est un planificateur non-linéaire développé sur la base d'une logique temporelle d'instants et d'intervalles restreinte, qui gère quasi-linéairement un treillis d'instants pour l'ajout de nouvelles contraintes. Le formalisme de représentation employé comporte deux niveaux de granularité : le niveau action et le niveau tâche. L'action est l'opérateur de plus bas niveau, elle est directement exécutable ; la tâche est composée, quant à elle, d'actions et de sous-tâches. La représentation temporelle utilisée permet de représenter des actions et des tâches dont les pré-conditions (ou les effets) disparaissent (ou apparaissent) au fur et à mesure de leur réalisation ainsi que des événements. Le plan est représenté par un treillis temporel sur lequel sont portées les actions, leurs pré-conditions et effets. Avant toute planification, les actions et les tâches sont compilées par le système en des treillis de structures similaires au treillis qui doit représenter le plan. Leur insertion dans le plan courant lors de la planification est ainsi facilitée. L'évolution vers une prise en compte plus réaliste du temps s'est donc faite de l'hypothèse de transitions instantanées vers l'utilisation de logiques temporelles d'instants ou d'intervalles permettant la modélisation d'actions avec recouvrements complexes.

Le plan initial se présente sous la forme d'une tâche, comprenant à la fois la définition complète de l'état du monde initial ainsi que les différents buts à satisfaire. Le monde initial est spécifié par des événements expliqués à l'instant t_{start} de la tâche initiale. La fonction de planification de IxTeT[22] est représenté dans l'algorithme 6.

```
Fonction Planifier(P,T)
début
    si P ne contient plus de défaut alors
        Retourner P
    sinon
        Sélection d'un défaut D de P
        Calcul de l'ensemble des résolvantes de D
        si cet ensemble est vide alors
            Retourner un échec
        sinon
            Choix d'une résolvante p
            Retourner (Planifier(P+p,T))
        fin
    fin
fin
```

Algorithme 6 – L'algorithme de IxTeT. P désigne le plan partiel et T désigne l'ensemble des modèles de tâches

[22] La boucle principale est basée sur un algorithme A_ϵ, variante de l'algorithme A*.

Le principe de l'algorithme ainsi que les différentes heuristiques employées sont détaillés dans [Laruelle 94].

IxTeT s'insère dans l'architecture globale d'un système de décision pour un robot autonome et constitue un système de raisonnement "hors niveau", dont le but est de fournir un plan de tâches calculé hors-ligne. Le plan produit est passé au module de supervision de tâches qui est chargé de l'affiner et, éventuellement, de modifier la tâche en cours en réaction à des événements imprévus. En cas de dysfonctionnement important (tâche échouée, ressources épuisées, arrivée d'événements imprévisibles, etc.), le superviseur peut remonter l'information au niveau du planificateur pour lancer une re-planification partielle de la mission. L'architecture de IxTeT, représentée dans la figure 3.2, est formée de trois composantes :

1. **Le module d'analyse** gère à la fois la recherche des résolvantes des sous-buts en cours (étape de faisabilité), l'identification des conflits induits et la recherche des résolvantes existant pour ces conflits (étape de satisfaisabilité).
2. **Le module de contrôle de la recherche** effectue une résolution globale selon trois niveaux :
   - choix d'une opération (sous-buts ou conflit),
   - choix de l'opérande, c'est-à-dire le sous-but ou le conflit particulier,
   - choix de la résolvante dans l'arbre de recherche globale.
3. **Le module de gestion du plan partiel** met à jour l'ensemble des prédicats, la table des variables et le graphe d'instants.

Ce planificateur a depuis été enrichi ; Laborie [Laborie 95] a intégré dans le planificateur un module de gestion de ressources permettant de prendre également en compte au niveau de contrôle de la recherche les conflits de ressources. Dans [Vidal 95], une extension concernant la gestion de l'incertain dans le temps a été entreprise. La représentation a ensuite été employée pour le suivi d'évolutions et la reconnaissance de chroniques. Dans [Laruelle 94], Laruelle propose la définition des différents algorithmes et des différentes heuristiques utilisées dans la représentation d'IxTeT. L'utilisation des ressources ainsi qu'une recherche hiérarchique a été introduite dans le planificateur [Laborie 95]. Dans [Gaborit 96], des outils de planifications supplémentaires comme la fusion et l'union de plans ont été introduits dans le planificateur en vue de l'application aux multi-robots. Albers [Albers 97] a enrichi la représentation d'IxTeT afin de prendre en compte dans la planification des tâches avec effets dépendant du contexte, (pour améliorer l'expressivité des actions, mais également pour améliorer les performances de la planification) et d'utiliser des axiomes de cohérence lors de l'élaboration de plans.

Le planificateur fait deux hypothèses restrictives qui correspondent à un compromis entre performance et richesse de représentation : il s'agit des hypothèses de complétude de la description du monde et du déterminisme des actions.

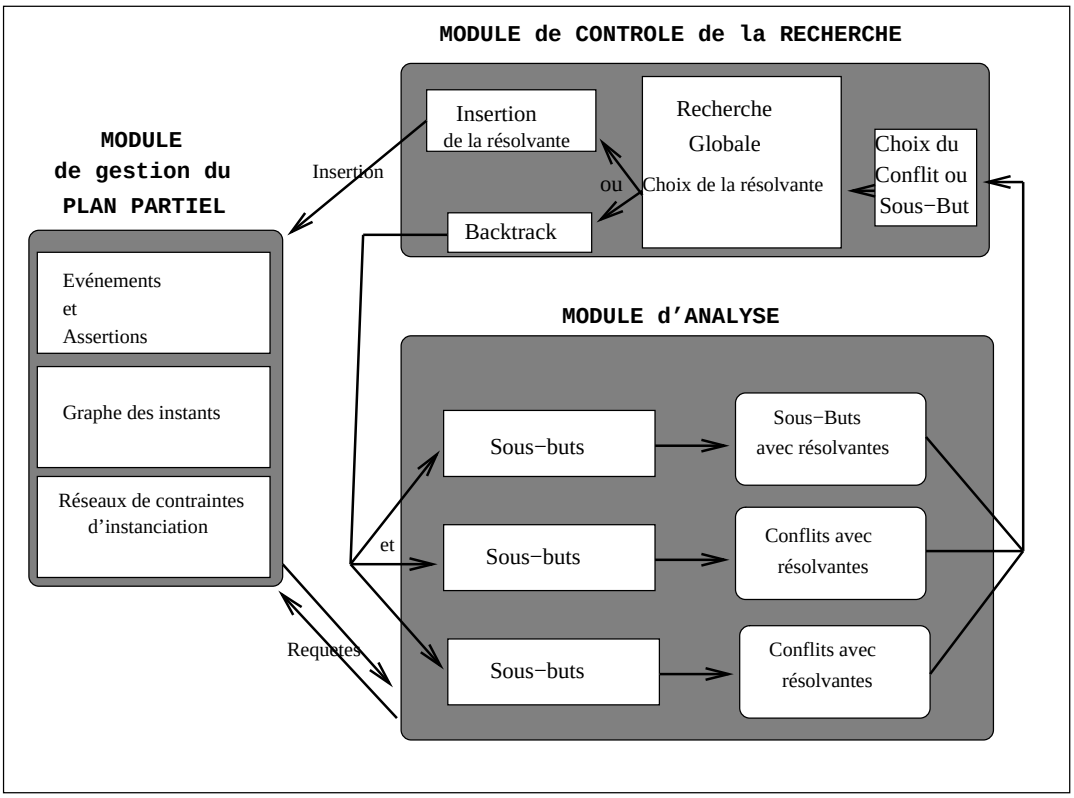

FIG. 3.2 – L'architecture générale du planificateur IxTeT (d'après [VIDAL 95])

Une extension d'IxTeT concernant l'exécution du plan a été élaborée dans le planificateur IxTeT-eXeC [LEMAI & INGRAND 04]. Cette approche intègre planification délibérative, réparation de plan et contrôle d'exécution dans un environnement dynamique avec des contraintes temporelles. Elle est basée sur deux composants : un planificateur et un exécutif qui tiennent compte explicitement du temps. Le planificateur produit un plan complet. Ce plan est ensuite déroulé par l'exécutif temporel selon un cycle "perception-réparation de plan/action" : intégrer les messages externes, réparer le plan si nécessaire, décider des actions à exécuter. Ce cycle permet d'être réactif aux échecs et dépassements de délai du système contrôlé et d'adapter le plan en conséquence.

Le système IxTeT (figure 3.3) comprend un planificateur capable de produire un plan, de replanifier dynamiquement et réparer un plan. Il est composé également d'un exécutif temporel. Les deux composants partagent le même plan au travers duquel ils interagissent. L'exécutif commence l'exécution d'un premier plan tant que celui-ci est valable. Si une situation non nominale se produit et qu'une réparation de plan est possible, elle est toujours tentée. Les avantages majeurs de la réparation sont de pouvoir continuer l'exécution des parties du plan encore valide, et a priori de nécessiter moins de décisions que la planification et donc d'être plus rapide. Finalement, si

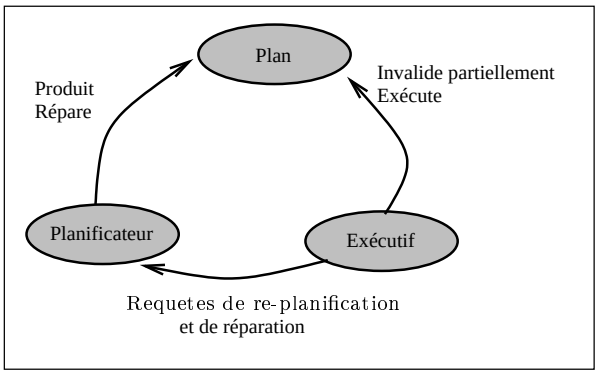

FIG. 3.3 – Schéma montrant l'organisation d'IxTeT

un plan ne peut être ni réparé, ni exécuté, une requête de planification sera alors envoyée au planificateur. L'exécutif contrôle la re-planification afin de garantir qu'il restera suffisamment de temps pour exécuter le nouveau plan.

3.3 CPT

CPT[23] [VIDAL & GEFFNER 04A, VIDAL & GEFFNER 04B] est un planificateur temporel, optimal et indépendant du domaine. Il est fondé sur la programmation par contraintes qui intègre les heuristiques existantes avec une nouvelle représentation et des règles de propagation qui parviennent à élaguer considérablement l'espace de recherche (éliminer des solutions partielles). La formulation proposée combine avec succès les possibilités d'exploration de l'espace de recherche de la planification dans les espaces de plans partiels avec des règles d'élagage puissantes et saines. La principale innovation de cette formulation est la capacité de raisonner autour des supports, des relations de précédence et des liens causaux, en faisant intervenir les actions qui n'appartiennent pas encore a un plan partiel. La formulation exploite la restriction de canonicité[24] interdisant à une action d'un domaine d'être présente plus d'une fois dans un plan. Cette restriction permet de regrouper les notions d'action et d'occurrence d'action, conduisant à plusieurs implications. L'objectif de CPT est de trouver des plans ayant une durée d'exécution minimale. Quand toutes les actions ont une durée uniforme, ce modèle se réduit au modèle standard de la planification parallèle.

CPT utilise la méthode de branchement dans la planification POCL[25] qui procède en choisis-

[23] Constraint Programming Temporal planner.

[24] Les plans canoniques sont à mi-chemin entre la planification temporelle classique et l'ordonnancement de tâches : dans ce dernier chaque action est exécutée exactement une fois, en planification canonique elle est exécutée au plus une fois, et en planification temporelle classique, elle peut être exécutée un nombre quelconque de fois.

[25] Partiel Order Causal Link.

sant un "défaut" (pré-condition ouverte ou menace) et en essayant chacune des réparations possibles [Weld 94]. Un état ou plan partiel dans l'espace de recherche correspond à un ensemble d'engagements représentés par un ensemble d'actions du plan, un ensemble de contraintes de précédence, un ensemble de liens causaux et un ensemble de pré-conditions ouvertes. Un état est terminal s'il est inconsistant (c.-à-d. que l'ordre de l'ensemble des contraintes de précédence est inconsistant ou cet état contient un défaut qui ne peut être réparé) ou s'il est un état but (il est consistant et ne contient plus de défaut).

L'adaptation du branchement POCL au cadre temporel est assez direct : il suffit d'ajouter de variables temporelles $T(a)$ représentant le temps initial d'une action a appartenant à l'ensemble d'actions. Ces variables temporelles ont comme domaine initial $T(Debut) = 0$, $T(Fin) = B$, et $T(a) :: [0; B - dur(a)]$ où B est la borne supérieure sur la durée d'exécution du plan ($Debut$ et Fin sont les deux "fausses" actions utilisées en planification POCL).

La formulation de la programmation par contraintes CP[26] de base du planificateur CPT est composée de quatre parties : pré-traitement, variables, contraintes et branchement. Après le pré-traitement, les variables sont créées et les contraintes postées et propagées. Si une inconsistance est trouvée, aucun plan valide[27] n'existe pour le problème. Sinon, la contrainte $T(Fin) = B$ pour la borne B initialisée au temps initial possible minimal de l'action Fin est postée et propagée. Les règles de branchement prennent alors le contrôle, et si aucune solution n'est trouvée, le processus recommence en rétractant la contrainte $T(Fin) = B$ et en la remplaçant par $T(Fin) = B + 1$, et ainsi de suite.

Les expérimentations montrent que ce planificateur est beaucoup plus performant que les planificateurs temporels optimaux existants. Il est également compétitif avec les meilleurs planificateurs parallèles dans le cas particulier où toutes les actions ont la même durée. Il a eu le deuxième prix de planificateurs optimaux à la quatrième compétition internationale de planification (IPC-04).

3.4 Autres planificateurs temporels

Dans cette section, nous présentons d'autres planificateurs temporels se trouvant dans la littérature du domaine de la planification.

– TGP[28] [Smith & Weld 99] est une extension de GraphPlan pour utiliser des actions ayant une durée. Seule la durée de l'action est prise en compte, les pré-conditions devant être vérifiées durant toute la durée d'exécution de l'action et les effets étant effectifs à la

[26] Constraint Programming.

[27] un plan valide est un plan dans lequel les exécutions des actions interférences ne se recouvrent pas dans le temps.

[28] Temporal GraphPlan.

fin de l'exécution de l'action. Par contre, la structure du graphe est modifiée pour prendre en compte la durée des actions par l'utilisation d'une structure circulaire à deux niveaux. Le niveau de départ d'un effet ou d'une action ne correspond pas au niveau d'apparition dans le graphe de planification classique, mais au temps minimum d'apparition calculé d'après la durée des actions. De plus, la notion d'exclusion mutuelle est étendue à l'exclusion entre actions et effets, afin de permettre le chevauchement de l'exécution des actions. Grâce à ce raisonnement sur les relations exclusivement mutuelles, TGP présente de bonnes performances par rapport à une version de GraphPlan qui ne gère pas les durées des actions.

– ParcPlan [Lever & Richards 94] résout des problèmes de planification, d'ordonnancement et de gestion de ressources tout en respectant les contraintes temporelles et en minimisant le coût. Ce planificateur utilise la notion d'intervalle ([début,fin)) pour représenter le temps. Chaque action est définie par une liste de conditions, une liste d'effets, une liste de ressources disponibles, une liste de contraintes et un coût. Le coût d'un plan est lié à la quantité de ressources utilisées, au coût d'exécution de chacune de ses tâches, au nombre de changements effectués pour établir le plan et aux préférences utilisées dans le choix du plan. L'algorithme de ParcPlan, qui commence en se basant sur une base de connaissances et un but, est composé de trois étapes :

  1. Sélectionner un but qui n'est pas encore unifié dans la base de données courante et chercher une action qui, grâce à son exécution, satisfait le but, c'est-à-dire que ce but peut être unifié avec un des effets de l'action. Ensuite :
     – introduire l'action dans le plan en unifiant le but et l'effet,
     – ajouter les effets de l'action à la base de données courante,
     – ajouter les conditions du but courant et supprimer le but sélectionné,
     – affirmer les contraintes,
     – évaluer le coût et ajouter le résultat au coût global.

     Cette étape continue avec les nouvelles valeurs de la base de données, les nouveaux buts et le coût global. La deuxième étape commence quand tous les buts sont unifiés dans la base de données.

  2. Cette étape a pour objectif d'unifier simultanément tous les buts avec des éléments de la base de données. Elle est considérée comme un problème de satisfaction de contraintes où chaque but est associé à un ensemble fini de variables qui décrit la représentation possible des buts dans la base de données et où les contraintes sont la possibilité d'unifier les buts aux éléments de la base de données.

  3. Après l'unification des buts avec la base de données, il est possible de trouver des variables, qui représentent les ressources ou le temps, ne sont pas encore instanciées.

En dernière étape, on cherche des valeurs de ces variables de façon à minimiser la fonction du coût.

Même si le planificateur ne réussit pas à exécuter les deux dernières étapes, il retourne au moins le résultat de la première et dans ce cas on trouve un plan qui contient des buts et des actions mais on ne sait pas l'ordre d'exécution de celles-ci. Dans ParcPlan, l'ordre de l'exécution des actions dans un plan est indépendant de l'ordre de leur introduction dans ce plan durant la recherche (planification non linéaire).

– Zeno [Penberthy & Weld 94] utilise la méthode de recherche dans l'espace de plans avec le moindre engagement pour engendrer un plan. Chaque action est définie par une liste de pré-conditions, une liste d'effets, un ensemble de contraintes qui définit son intervalle de temps et ses prédécesseurs et un ensemble de ressources limitées. Les buts sont contraints par des dates-limites (deadline). L'algorithme de Zeno commence par un nœud qui forme le plan et qui contient un agenda du but. À chaque nœud, il enlève de l'agenda du but les éléments unifiés avec le nœud. Il se termine quand il trouve un nœud où l'agenda du but est vide (plan-solution) ou quand il trouve que les contraintes de plan sont consistantes (échec). Cet algorithme, démontré sain et complet, est composé de trois étapes :

1. décomposer les buts compliqués en sous-buts,
2. choisir les actions qui satisfont les buts simplifiés,
3. introduire les contraintes pour vérifier l'interaction entre les actions et les buts.

Afin d'obtenir la complétude de cet algorithme, il était nécessaire de faire des retours en arrière pour ré-exploiter des nœuds et vérifier les contraintes. Le facteur de branchement est proportionnel au nombre des actions disponibles et au nombre de buts disjonctifs.

– Sapa [Do & Kambhampati 03] utilise la méthode de recherche dans l'espace d'états A^* en chaînage avant pour exploiter les actions paramétrées par des durées et une consommation continue de ressources. À chaque action est associée une liste de pré-conditions, une liste des effets, une date de début, une date de fin, une durée et un coût. L'objectif de Sapa est de produire un plan avec un nombre minimum d'actions et un coût minimal.
Le principe de l'algorithme de recherche (algorithme 7) est le suivant : la recherche commence par l'état initial et à chaque fois qu'il trouve une action A applicable à l'état courant S, il l'ajoute à l'état courant qu'il met dans la queue (fonction Enqueue). La fonction Dequeue est utilisée pour enlever de la queue le premier état. La queue d'états dépend d'une fonction heuristique qui mesure la difficulté d'atteindre les buts à partir de l'état courant. Sapa traite le problème de planification multi-critères : il cherche un plan qui satisfait les contraintes temporelles (date de début et de fin de chaque tâche, date limite de fin du

```
Queue : Q = {S_init}
début
    tant que Q ≠ {} faire
        S :=Dequeue(Q) /*Retirer le premier élément de file*/
        Sélectionner A non-déterministe applicable en S
        S' :=appliquer(A,S)
        si S' satisfait G alors
            solution
        sinon
            Enqueue(S',Q) /*Ajout à la fin de la file*/
        fin
    fin
fin
```

Algorithme 7 – L'algorithme de recherche de Sapa

plan complet) et qui a un coût d'exécution minimal. Sapa était un des meilleurs planificateurs traitant les contraintes temporelles dans la troisième compétition internationale de planification AIPS-02.

3.5 Limites de la planification temporelle

Un planificateur temporel produit un plan qui respecte les contraintes temporelles et qui peut s'exécuter avant la "deadline". Dans ce chapitre, nous avons présenté les planificateurs qui prennent en compte le temps. Certains d'entre s'intéressent aux contraintes temporelles qualitatives, c.-à-d. les contraintes de précédence ; d'autres s'intéressent aux contraintes quantitatives, c.-à-d. les dates de début, les dates de fin et les durées. Seulement un petit nombre d'entre eux utilisent ces deux types à la fois. Les planificateurs temporels ajoutent la notion de temps aux planificateurs classiques, mais sont applicables seulement dans un environnement statique où les actions sont déterministes et où tout est observable.

Dans le chapitre suivant, nous traitons les planificateurs qui permettent de prendre en compte un autre critère d'évaluation par rapport aux planificateurs classiques : l'incertitude.

Chapitre 4

La planification sous incertitude

Même si la représentation du monde est complète, il peut arriver que des événements imprévus surviennent en cours d'exécution de plan, alors qu'ils n'étaient pas prévus dans la planification. L'incertain en planification joue un rôle important dès que l'on se trouve confronté à des problèmes réels.

La planification classique fait de nombreuses hypothèses simplificatrices dont certaines sont très fortes. Pollack et Horty [Pollack & Horty 99] font ressortir six hypothèses faites par les techniques de planification classiques :

- L'agent est omniscient, il connaît tout de son environnement.
- Les actions peuvent être exécutées par l'agent et ont des conséquences déterministes.
- Pas de notions d'atteinte partielle des objectifs.
- L'agent est la seule source de changement dans son environnement ; le monde est statique.
- Les objectifs de l'agent sont fixes tout au long de la planification et de l'exécution du plan.
- Les actions faites par l'agent ont un effet instantané.

Les techniques de planification en incertitude permettent de lever certaines de ces hypothèses, proposant ainsi des solutions de planification applicables à des problèmes plus réalistes que ceux de la planification classique. Parmi les approches proposées dans la littérature, on cite la planification conditionnelle, la planification probabiliste, la planification conditionnelle probabiliste, la planification temporelle probabiliste et le processus de décision markoviens.

4.1 La planification conditionnelle

La procédure de planification conditionnelle génère un plan dans lequel la séquence d'actions peut s'adapter aux observations faites par l'agent lors de l'exécution du plan. Cette technique peut tenir compte d'une incertitude aussi bien au niveau de l'état de l'environnement que des effets de certaines actions. Elle génère un plan qui tient compte de toutes les possibilités. Il s'agit d'un plan sous forme d'arbre où chaque changement de branche représente les observations

devant être faites lors de l'exécution. La figure 4.1 représente la procédure de la planification conditionnelle. Warplan-C [Warren 76], développé en 1976, est le premier planificateur conditionnel. Il crée des plans comme ceux de la figure 4.1 en limitant les actions d'observation à deux : une proposition p et sa négation $\sim p$. Le planificateur conditionnel CNLP[29] [Peot & Smith 92], apparu en 1992, est basé sur les opérateurs STRIPS, mais ne limite pas l'expression des résultats des actions d'observations comme dans le cas de Warplan-C. Un point négatif de ce planificateur est que la taille des plans produits est soumise à une croissance exponentielle en nombre d'actions d'observation.
Ce problème de croissance exponentielle fait de la planification conditionnelle une stratégie difficilement utilisable. Cependant, elle est intéressante dans la mesure où elle permet le développement d'une planification plus réaliste si l'agent connaît son environnement de façon moins complète.

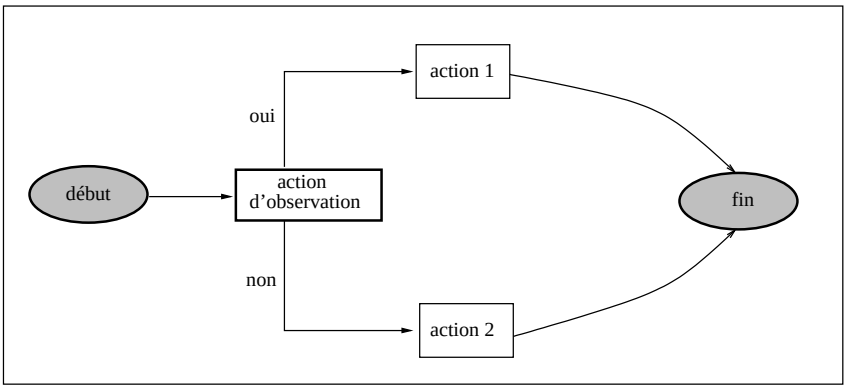

Fig. 4.1 – La procédure de la planification conditionnelle

4.2 La planification probabiliste

La planification probabiliste produit un plan ayant de fortes chances de réussir, c'est-à-dire un plan dont la probabilité de succès est très forte ou est supérieure à un seuil quelconque. Dans une telle planification, l'information sur les états du monde n'a à être ni complète, ni déterministe. De plus, l'agent n'a pas à connaître parfaitement les effets de ses actions. Nous présentons les principales approches proposées dans la littérature en faisant ressortir leurs forces et leurs faiblesses.

4.2.1 BURIDAN

Kushmerick, Hanks et Weld [Kushmerick *et al.* 94, Kushmerick *et al.* 95] présentent le planificateur probabiliste BURIDAN basé lui aussi sur la représentation STRIPS. Les opérateurs

[29] Conditional Nonlinear Planner.

sont représentés par des arbres binaires dans lesquels les feuilles correspondent aux conséquences possibles de l'action qui est à la racine de l'arbre. Les nœuds internes décrivent l'état du monde permettant d'obtenir l'effet défini dans la feuille située au bout de sa branche et une probabilité est associée à chaque transition vers une feuille. Les chemins pour tous les effets de l'action doivent être mutuellement exclusifs et exhaustifs, c'est-à-dire qu'exactement une conséquence sera réalisée à la suite de l'exécution de l'action. La figure 4.2 illustre un exemple de l'opérateur probabiliste dans BURIDAN. L'action principale peut être exécutée dans deux cas : si l'action d'observation est vraie ou fausse (par exemple un robot peut prendre un bloc si son bras est sec ou mouillé). Pour chacune de ces deux observations, il y a deux effets selon si l'action principale est réussie ou non. La somme des probabilités d'une pré-condition vers ses deux effets est égale à 1 ($p1 + p2 = 1, p3 + p4 = 1$).

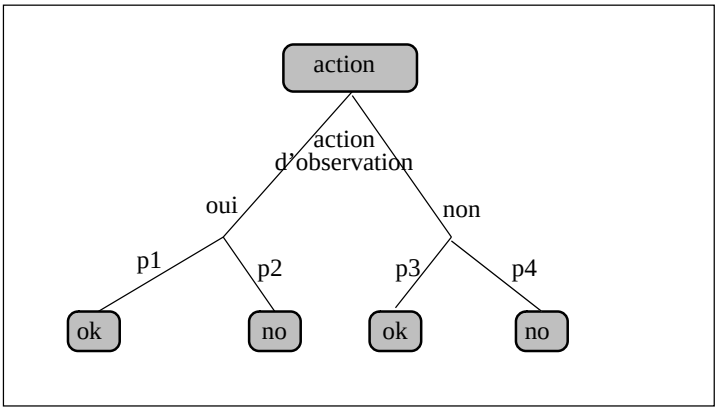

FIG. 4.2 – Un exemple d'un opérateur probabiliste dans BURIDAN

Les états du monde sont représentés par des distributions de probabilités sur les valeurs des variables qui les définissent. L'objectif du planificateur probabiliste est de générer une séquence d'actions permettant de passer d'une distribution de probabilités initiale vers une autre distribution dans laquelle le but à atteindre a de fortes chances de se réaliser. La planification débute avec un plan contenant la distribution de probabilités initiale et l'état final à atteindre et se fait par une recherche dans l'espace des plans partiels. On vérifie d'abord si la probabilité que l'objectif soit atteint est supérieure au seuil au-delà duquel un plan est acceptable. Si oui, la planification est terminée. Autrement, le planificateur cherche à augmenter cette probabilité en raffinant le plan, c.-à-d. en ajoutant des opérateurs ou en résolvant des conflits dans le but d'augmenter la probabilité de succès du plan. L'algorithme de BURIDAN est démontré sain et complet.

4.2.2 PGraphPlan et TGraphPlan

Plusieurs extensions de GraphPlan ont été proposées pour gérer l'incertitude sur les actions en environnement totalement ou partiellement observable. PGraphPlan et TGraphPlan, comme leurs noms l'indiquent, sont des extensions de GraphPlan ajoutant la notion de probabilité.

La différence entre PGraphPlan [Blum & Langford 99] et GraphPlan (cf. section 2.6.2) est que le premier utilise la recherche en avant dans l'algorithme de recherche au lieu de la recherche en arrière utilisée dans le deuxième, et de plus ne permet pas l'exécution simultanée des actions. La construction du graphe de planification se fait de la même façon qu'en GraphPlan sauf que chaque action contient un ensemble de possibilités, à chacune desquelles est associée une probabilité. Les arcs sortant de l'action indiquent ses différentes possibilités. Supposons qu'un opérateur peut exécuter les deux effets *ajouter p à la liste* et *supprimer q de la liste* avec une probabilité 0.7 et l'effet *ajouter p à la liste* avec une probabilité 0.3. Dans ce cas il y a deux possibilités avec des probabilités 0.7 et 0.3. Mais il y aura trois arcs sortant de l'action : le premier pour indiquer l'effet *ajouter p à la liste* associé à la première possibilité, le deuxième pour indiquer l'effet *supprimer q de la liste* associé à la première possibilité et le troisième pour indiquer l'effet *ajouter p à la liste* associé à la deuxième possibilité. PGraphPlan produit un plan optimal, mais l'ajout des effets probabilistes a diminué sa performance (en terme de temps) par rapport à GraphPlan. Il reste quand même plus performant que plusieurs autres planificateurs probabilistes [Blum & Langford 99].

L'autre extension probabiliste de GraphPlan est le planificateur TGraphPlan [Blum & Langford 99] qui, comme GraphPlan, utilise la méthode de recherche en arrière pour trouver un plan solution. La performance de TGraphPlan est la même que GraphPlan mais, il ne fournit pas le plan optimal puisque dans chaque étape, il choisit l'action qui a la plus grande probabilité. La probabilité totale d'un plan est calculée en multipliant la probabilité de l'action qui a la plus grande probabilité dans l'étape courante avec la probabilité du plan partiel déjà construit.

La performance de TGraphPlan est plus grande que PGraphPlan. Une comparaison entre ces deux planificateurs est faite dans [Blum & Langford 99].

Une autre extension de GraphPlan concernant l'incertitude est le planificateur CGP[30] [Smith & Weld 98] qui permet de gérer l'incertitude dans les effets des actions (effets "non-déterministes") ainsi que la connaissance partielle de l'état initial. Plusieurs graphes de planification sont créés pour toutes les possibilités correspondant à chaque source d'incertitude, ce qui induit une structure très lourde et des performances peu efficaces.

[30] Conformant GraphPlan.

4.2.3 Autres planificateurs probabilistes

Dans cette section, nous présentons autres planificateurs probabilistes se trouvant dans la littérature du domaine de la planification.

- MaxPlan [Majercik & Littman 98] est un planificateur probabiliste dans un domaine non observable (les effets des actions précédentes ne peuvent pas être utilisés dans la sélection d'une action courante). Ce domaine est caractérisé par un ensemble fini d'états, un ensemble fini d'actions (ou opérations), un état initial et un ensemble de buts. L'objectif est de choisir des actions, l'une après l'autre, pour aller de l'état initial à un des états finaux avec une probabilité dépassant un seuil quelconque. L'algorithme de MaxPlan est divisé en deux phases :

  1. Convertir le problème de planification en une instance d'E-MAJSAT : un problème E-MAJSAT [Littman 99] est un problème de planification probabiliste $NP^{PP}-$*complet*. Le convertisseur sélectionne un horizon du plan N, indexe en temps chaque proposition et chaque action pour que le planificateur puisse raisonner à propos de "quoi" se produit "quand", et ensuite il vérifie la satisfaisabilité en respectant les conditions suivantes :
     - Les conditions initiales sont réalisées au temps 0 et les conditions du but au temps N.
     - Les actions au temps t sont mutuellement exclusive ($1 \leq t \leq N$).
     - Une proposition p est *vraie* au temps t si elle était vraie au temps $t-1$ et si l'action exécutée au temps t ne l'a pas rendue *fausse*, ou si l'action au temps t l'a rendue *vraie* ($1 \leq t \leq N$).
  2. Résoudre cette instance en appliquant les techniques de la programmation dynamique et de la satisfaisabilité booléenne : la résolution du problème converti en E-MAJSAT produit un plan avec une grande probabilité de succès qui dépasse un seuil donné. L'idée est de construire un arbre binaire de variables où chaque nœud représente une variable de choix (la valeur vraie est arbitraire) ou de chance (la valeur vraie est déterminée par un ensemble de probabilités), et chaque sous-arbre représente les deux résultats possibles de chaque nœud parent. La probabilité des branches dépend du nœud selon qu'il est un choix ou une chance.

  MaxPlan essaie d'explorer seulement l'espace de recherche nécessaire pour produire le plan avec la probabilité demandée. Cela évite d'augmenter exponentiellement la taille de l'espace de recherche et l'horizon du plan.

- Probapop [Onder *et al.* 04, Onder *et al.* 06] est un planificateur d'ordre partiel tels que les actions et l'état initial sont probabilistes et l'environnement n'est pas observable.

L'objectif est de trouver le nombre d'étapes minimal amenant un agent d'un état initial à un état final avec une probabilité qui dépasse un seuil donné. Probapop construit d'abord un plan initial qui amène de l'état initial à l'état but ; ensuite il le raffine jusqu'à ce qu'il trouve la probabilité cherchée ou qu'il la dépasse. Si plusieurs plans ont la même probabilité, il choisit celui qui a le nombre d'étapes le plus petit. Probapop, qui fournit un plan sain et complet, a participé à la première compétition internationale de planification probabiliste (IPC-04) et a montré son efficacité dans la résolution de plusieurs applications probabilistes.

4.3 La planification conditionnelle probabiliste

La planification conditionnelle et la planification probabiliste ont des avantages lorsque l'agent travaille dans un environnement incertain. La planification conditionnelle permet à l'agent de choisir la branche de l'arbre correspond à l'état du monde qu'il observe lors de l'exécution. La planification probabiliste permet à l'agent de générer un plan qui a une très grande probabilité d'être exécuté correctement lorsque l'agent n'a pas une observation certaine de l'état du monde. Ces deux approches ont des hypothèses opposées. La première approche suppose que l'agent peut observer l'état de son environnement lors de l'exécution de son plan mais n'a pas une distribution de probabilités des variables de son environnement ; par contre la deuxième approche suppose que l'agent a une distribution de probabilités de variables de son environnement, mais ne peut pas observer l'état de son environnement lors de l'exécution de son plan. Une combinaison de ces deux approches a été proposée dans la littérature, notamment dans les planificateurs C-BURIDAN et Mahinur que nous détaillons dans les deux sections suivantes.

4.3.1 C-BURIDAN

C-BURIDAN [Draper *et al.* 94], basé sur BURIDAN [Kushmerick *et al.* 94, Kushmerick *et al.* 95] (cf. Section 4.2.1) et CNLP [Peot & Smith 92] (cf. Section 4.1), est un planificateur conditionnel probabiliste qui construit un plan d'actions probabilistes (productrices d'information) et d'exécution contingente. Le planificateur prend en entrée une distribution de probabilités sur les états initiaux, un ensemble d'actions, un but et un seuil de probabilité et produit un plan contingent qui satisfait le but avec une probabilité dépassant ou égale au seuil. Ce planificateur utilise une sonde (sensor) pour observer les états du monde. La représentation de l'action en C-BURIDAN est basée sur la distribution de probabilités donnée et le rapport retourné par la sonde. Un état, qui est une description complète du monde d'agent à un moment donné, contient un ensemble de propositions ayant des valeurs *vraie* ou *fausse* selon le résultat de l'observateur. La base de l'algorithme de calcul des plans est la suivante :

1. Commencer par le plan initial qui contient dans l'ordre l'action initiale et le but.
2. Itérer :
   - Évaluer le plan : calculer la probabilité dont le plan courant achève le but. Rapporter Succès si cette probabilité est au moins égale au seuil.
   - Sinon choisir d'une façon non déterministe un raffinement pour le plan courant. Rapporter Échec si aucun raffinement n'est possible. Sinon appliquer le raffinement au plan courant et répéter.

L'algorithme calcule les probabilités des états générés par chaque action et fait la somme de toutes les probabilités des états finaux qui ont satisfait le but. Si le résultat de la somme est supérieure ou égal au seuil, l'algorithme retourne la séquence d'actions ; sinon il continue avec des nouvelles actions. Si aucune séquence d'actions n'a une probabilité supérieur ou égale au seuil, il retourne un échec. L'algorithme est démontré sain et complet, mais ses solutions sont satisfaisantes plutôt qu'optimales.

4.3.2 Mahinur

Mahinur [Onder & Pollack 97, Onder & Pollack 99] est un planificateur conditionnel probabiliste qui utilise la recherche dans l'espace des plans où les nœuds d'un espace de recherche représentent les plans partiels. L'objectif est de trouver un plan avec la probabilité de succès maximale. L'entrée de l'algorithme est un ensemble de conditions initiales, un ensemble de conditions de but et un temps limite. La sortie est un ensemble de plans. L'algorithme est divisé en deux étapes principales :

1. Construire un plan squelette.
2. Raffiner le plan : tant que le plan est en dessous d'un seuil donné :
   (a) Sélectionner l'éventualité dont l'échec a la dis-utilité[31] maximale.
   (b) Étendre le plan pour inclure des actions afin de corriger l'échec.

D'abord, l'algorithme trouve le plan squelette, c.-à-d. un plan minimal dont la probabilité de succès est différente de 0. Durant chaque itération, une éventualité (un sous-ensemble de résultats d'une action probabiliste) dont l'échec a la dis-utilité[31] maximale est choisie. Ensuite, le plan est étendu pour inclure des actions qui corrigent l'échec trouvé. La boucle est répétée jusqu'à ce que la probabilité du plan atteigne un seuil donné ou jusqu'à ce que le temps de planification achève.

4.4 La planification temporelle probabiliste

La planification temporelle probabiliste est la combinaison de la planification temporelle et la planification probabiliste. Dans cette planification, les actions ont des contraintes temporelles

[31] La probabilité d'échec d'une éventualité multipliée par la somme des valeurs de tous les buts qu'elle supporte.

(durées d'exécution, date de début, date de fin, précédence) et les effets sont probabilistes. Cette approche est la plus récente dans le domaine de planification en IA puisque les planificateurs combinant les deux aspects n'ont commencé qu'en fin 2003.

4.4.1 Planification d'un voyage

Dans cette section, nous présentons un planificateur qui propose une solution à un problème bien précis nécessitant la gestion des préférences de l'agent et d'un environnement incertain : la planification de voyages [Bérubé 03]. Ce planificateur, basé sur des opérateurs STRIPS, est présenté dans la figure 4.3. L'algorithme de recherche utilisé est celui de la recherche dans l'espace des plans, plus précisément l'algorithme A^*. Il s'agit d'une suite d'itérations permettant de passer d'un plan minimal à un plan complet en raffinant des solutions partielles.

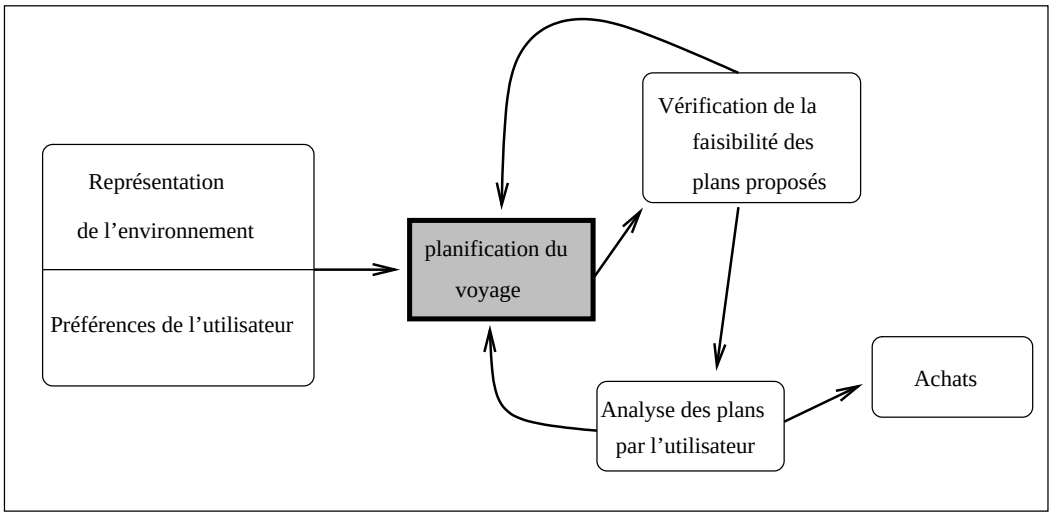

Fig. 4.3 – Le planificateur de voyages (d'après [Bérubé 03])

La différence avec les algorithmes classiques est que cet algorithme offre la possibilité de produire plusieurs plans alternatifs. Un utilisateur peut toujours demander qu'on lui présente un plan supplémentaire afin qu'il puisse ensuite choisir celui qu'il préfère. L'objectif de cet algorithme n'est donc pas de produire le meilleur plan mais bien de produire les meilleurs plans correspondant au problème à résoudre et basés sur les préférences de l'utilisateur. Ces préférences, représentées par une fonction d'utilité, portent sur trois critères : le coût, les contraintes temporelles et le confort.

L'algorithme de ce planificateur est le suivant :

1. Vérifier si le plan est complet avant de poursuivre.
2. Choisir le plan à raffiner (le meilleur plan partiel).
3. Choisir l'objectif à atteindre parmi les conditions à satisfaire de ce meilleur plan.

4. Vérifier s'il est possible de satisfaire l'objectif à partir des effets déjà présents dans le plan. Si oui, faire le lien avec cet effet et passer à l'étape 7.
5. Choisir, selon la fonction d'utilité, le meilleur opérateur permettant de satisfaire l'objectif.
6. Créer une nouvelle instance de plan partiel en copiant le plan à raffiner et en ajoutant l'opérateur choisi en 5 à cette copie.
7. Vérifier si le plan est complet. Si c'est le cas, le transférer de la liste des plans partiels vers l'ensemble des plans complets. Sinon, l'insérer dans la liste de plans partiels. Retourner à l'étape 1.

Les tests effectués sur ce planificateur montrent qu'au niveau de la qualité des solutions, l'algorithme utilisé est excellent puisqu'il trouve toujours les meilleures solutions qui s'adaptent aux préférences de l'utilisateur. Par contre, la complexité en temps et en espace mémoire est potentiellement très élevée. L'algorithme de recherche A^* exige que la taille de la solution soit relativement petite afin d'éviter que l'explosion combinatoire ne crée des problèmes d'espace mémoire et de temps de calcul.

D'un point de vue général, ce planificateur a beaucoup de points communs avec le nôtre : contraintes temporelles, coûts, incertitudes et une fonction d'utilité. Malgré cette ressemblance, son application précise à un seul problème de planification bien déterminé le limite et nous empêche de faire une vraie comparaison avec notre approche.

4.4.2 Planification des opérations militaires

L'objectif du planificateur d'opérations militaires [Aberdeen *et al.* 04] est d'assigner un ensemble de tâches et de ressources à une mission tout en minimisant le coût. Les étapes de planification dans ce domaine sont :

1. Analyse de mission : on décrit les opérations et les résultats désirés.
2. Développement de course-d'actions : on choisit des tâches qui peuvent dépendre des résultats opérationnels précédents et on établit des nouveaux résultats.
3. Analyse de course-d'actions : les faiblesses à la course-d'actions sont identifiées et corrigées.
4. Décision et exécution : le commandant choisit la course-d'actions et prend la décision.

Ce planificateur combine l'utilisation de plusieurs sortes de contraintes comme l'incertitude, les ressources et le coût. Chaque tâche a un ensemble de pré-conditions, un ensemble d'effets, un ensemble de ressources, une probabilité d'échec et une durée d'exécution. Une tâche est prête à s'exécuter si ses pré-conditions sont satisfaites et s'il y a assez de ressources pour son exécution. L'exécution d'une tâche fournit des effets négatifs ou positifs et diminue la quantité de ressources courante. Le domaine de planification est organisé sous la forme d'un arbre d'ordonnancement. Cet arbre décrit quelles tâches courantes doivent être exécutées en donnant l'historique des

opérations effectuées. Chaque branche de l'arbre est marquée par la probabilité nécessaire pour son exécution, la probabilité de réaliser le but, les valeurs actuelles des effets et de sous-buts et l'utilisation de ressources. L'algorithme de recherche des actions est le suivant :

```
Fonction trouver_Actions(État e)
Variables : Action a, État e', État nouveau_État, Liste_d_États successeurs, Tâche t
début
    pour chaque a = sous-ensemble de tâches exclusivement non mutuelles faire
        nouveau_État = e.copier()
        pour chaque t = tâche commencée par a faire
            nouveau_État.commencer_Tâche(t)
            nouveau_État.ajouter_Événement(fin de t)
        fin
        nouveau_État.prob = 1.0
        trouver_Successeur(nouveau_État,successeurs)
        e.ajouter_Action(a,successeurs)
    fin
fin
```

Algorithme 8 – L'algorithme de recherche des actions dans le planificateur d'opérations militaires

Ce planificateur utilise la méthode de recherche heuristique LRTDP[32] [Bonet & Geffner 03] qui explore incrémentalement l'espace de recherche et génère les plans en utilisant une fonction d'optimisation. Cette fonction a pour but de minimiser la probabilité d'échec, de minimiser la durée d'exécution et de minimiser le coût ou bien un seul de ces critères. La fonction trouver_Successeur détermine les états successeurs et leurs probabilités pour chaque action. Chaque action qui viole la limite de ressources ou qui passe la définition d'exclusivité mutuelle est supprimée de l'ensemble d'actions.

4.4.3 Prottle

Prottle [Little 04, Little *et al.* 05] est un planificateur temporel probabiliste qui utilise la recherche dans un graphe ET/OU, tel qu'un nœud ET représente une chance et un nœud OU représente un choix, pour résoudre le problème de planification. Dans ce graphe, les nœuds ET sont associés aux événements probabilistes alternatifs et les nœuds OU sont associés à la sélection des actions.

Chaque nœud est utilisé pour une sélection ou pour un avancement. Cette double fonction

[32] Labelled Real Time Dynamic Programming.

des nœuds est utilisée quand les états sont partitionnés entre ceux qui représentent la sélection d'actions et ceux qui représentent l'avancement en temps. Cette sorte d'optimisation permet aux planificateurs utilisant le chaînage avant d'être mieux guidés par les actions structurées en séquences linéaires. Les règles de succession de nœuds sont représentées dans la figure 4.4 : chaque successeur d'un nœud doit être un nœud de sélection du même type ou un nœud d'avancement d'un type opposé. Dans Prottle, un état de l'espace de recherche est défini par un nœud du graphe ET/OU identifié par *un temps*, *un modèle* et *une file d'événements*. Le temps d'un état est, en général, le même que ses prédécesseurs mais il augmente dans le cas où on avance d'un choix à une chance (les arcs en gras dans la figure 4.4). Le modèle est l'ensemble des valeurs *vraies* pour chacune des propositions et l'événement (un effet, un événement probabiliste ou une condition d'exécution d'une tâche qui demande une vérification) est une liste ordonnée des événements en suspens. L'état initial est défini par un état de choix d'avancement avec un temps 0, le modèle initial et une liste d'événements vide. Un état final est un état où le modèle satisfait le but du problème. À chaque état on associe une limite de coût inférieure et une limite de coût supérieure. Le coût d'un état est la probabilité que le but ne soit pas atteint de cet état que seulement avec des choix optimaux.

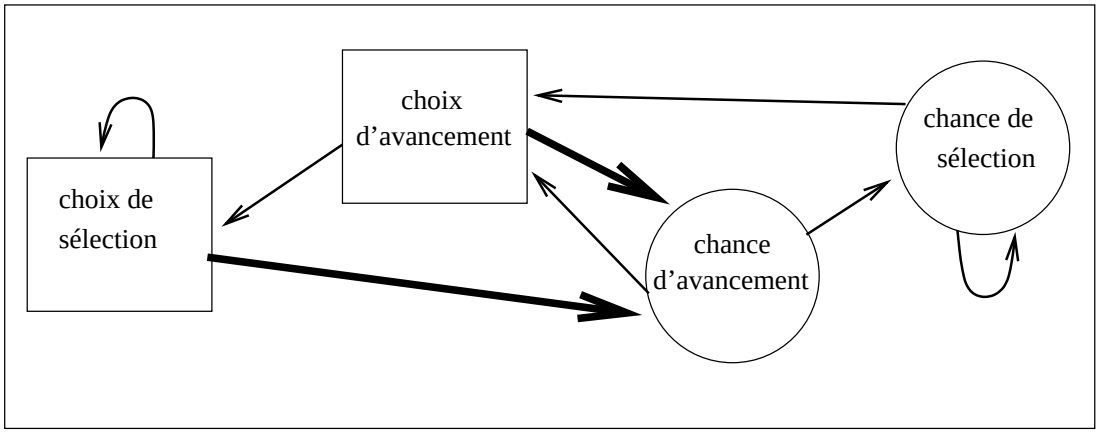

FIG. 4.4 – La succession des nœuds dans le système Prottle (d'après [LITTLE *et al.* 05])

L'algorithme de Prottle (représenté dans l'algorithme 9) explore l'espace de recherche par la méthode de recherche en profondeur d'abord en commençant par l'état initial. La fonction "Prouver" est répétée jusqu'à la résolution de l'état initial. Quand un état est sélectionné, ses successeurs sont crées et initialisés. Les limites de coût sont de la forme limite inférieure (lower bound) et limite supérieure (upper bound). Dans l'étape de recherche, la limite de coût inférieure augmente et la limite de coût supérieure diminue. Les nouveaux états ont une limite inférieure égale à 0 et une limite supérieure égale à 1 ou bien des valeurs calculées à partir des fonctions d'heuristiques. Les coûts d'un état sont mis à jour en comparant leurs valeurs actuelles avec

```
Rechercher(état-initial)
début
    répéter
        Prouver(état-initial)
    jusqu'à Étiquette(état-initial)=Résolu;
fin
Prouver(état)
début
    si Étiquette(état) ≠ Résolu alors
        si ¬ Augmenté(état) alors
            Augmenter(état)
        fin
        Prouver(Sélectionner-Successeur(état))
        Mettre-à-jour-Limites-de-Coût(état)
        Mettre-à-jour-Étiquette(état)
    fin
fin
```

Algorithme 9 – L'algorithme de recherche utilisé dans Prottle

celles de ses successeurs. Les formules suivantes sont utilisées pour mettre à jour les limites de coût d'un état :

$L_{choix}(s) := max(L(s), min_{s' \in S(s)} L(s'))$,

$U_{choix}(s) := min(U(s), min_{s' \in S(s)} U(s'))$,

$L_{chance}(s) := max(L(s), \sum_{s' \in S(s)} P(s')L(s'))$,

$U_{chance}(s) := min(U(s), \sum_{s' \in S(s)} P(s')U(s'))$.

Dans ces formules, s désigne un état, s' est son successeur, L(s) et U(s) sont les limites de coût inférieures et suppérieures d'un état s et P(s) est la probabilité de l'état s. Les limites inférieures et supérieures déterminent la fin de la recherche. Chaque étape avec un temps supérieur à un temps limite défini est un étape échoué. Le plan-solution est celui qui a la plus grande probabilité.

4.5 Processus décisionnel de Markov

Certaines classes de problèmes de planification peuvent être modélisées comme des processus décisionnels markoviens (MDPs) [Puterman 94]. C'est le cas, par exemple, des problèmes dans lesquels l'agent évolue dans un environnement dynamique pouvant être modélisé par un processus stochastique et où les actions peuvent influencer le comportement de ce processus. Ainsi, l'état dans lequel se trouve l'agent et la décision qu'il prend déterminent conjointement la distribution

de probabilités des prochains états.

Les MDPs permettent, à partir d'un état initial, d'effets sur les actions et d'objectifs, d'obtenir une politique d'action dont l'exécution parviendra au mieux à réaliser les objectifs de l'agent. Une politique d'action fait correspondre à chaque état une action. Lorsque l'agent est dans un état s, dire qu'il suit une politique Π signifie qu'il choisit d'effectuer l'action associée à s dans Π. À partir d'un état s, une action a peut mener l'agent dans différents états s'. Chaque transition de s à s' est caractérisée par une probabilité de transition notée $p(s'|s,a)$[33]. À chaque état s est associée une valeur (récompense) $V(s)$ afin de représenter l'intérêt que l'environnement soit dans cet état. À chaque action est associée une valeur qui représente ce que l'on espère gagner par l'application de cette action. Le gain espéré est calculé selon la formule suivante :

$$Q(a,s) = \sum_{s' \in S} P(s'|s,a) \cdot V(s')$$

L'objectif d'un MDP est de maximiser une fonction d'utilité calculée à partir des récompenses et du gain espéré. Pour cela, un MDP détermine la politique optimale qui associe à chaque état l'action qui maximise le gain espéré à long terme. Dans la littérature, il existe plusieurs méthodes pour résoudre un MDP. Nous présentons les deux plus connues *Policy Iteration* [Howard 60] et *Value Iteration* [Bellman 57].

4.5.1 L'algorithme de *Policy Iteration*

Cet algorithme est fondé sur le fait que la connaissance d'une politique Π et des valeurs $V(s), s \in S$ correspondant à Π, permet de trouver une politique Π' qui est meilleure que Π. De plus, l'amélioration progressive d'une politique conduit à la politique optimale. Le calcul d'une politique optimale selon cet algorithme se réalise en deux phases :

1. **L'évaluation :** l'évaluation d'une politique est le calcul des valeurs $V(s)$ pour tout $s \in S$ sachant que la politique Π est suivie dans l'état s et le sera dans les états futurs. La condition d'arrêt est que la différence entre l'ancienne valeur $V(s)$ et celle produite par l'itération actuelle reste inférieure ou égale à ϵ.
2. **L'amélioration :** cette phase a pour but d'améliorer la politique Π trouvée dans la première phase.

L'algorithme *Policy Iteration* est basé sur l'idée que la politique produite par l'amélioration d'une autre politique peut être à son tour améliorée, et ainsi de suite. Ce processus Évaluer-Améliorer s'arrête une fois que la politique actuelle ne s'améliore plus (elle est déjà optimale). L'algorithme de *Policy Iteration*, démontré polynomial, est représenté dans l'algorithme 10. Chaque itération

[33] C'est la probabilité que l'environnement soit dans l'état s' à l'instant $i+1$ sachant qu'à l'instant i il est dans s et que l'action a est appliquée.

Données : Un MDP fini, une politique initiale Π, un petit réel positif ϵ
Résultat : une politique optimale avec une marge d'erreur $< \epsilon$
début
  Initialisation $V(s) = 0, \forall s \in S$
  répéter
    $politique\_stable \leftarrow vrai$
    répéter
      $\Delta \leftarrow 0$
      pour tout $s \in S$ **faire**
        $v \leftarrow V(s)$
        $V(s) \leftarrow \alpha(s) + \gamma.\{\sum_{s' \in S} P(s'|s, \Pi(s)) \cdot V(s')\}$
        $\Delta \leftarrow max\{\Delta, |v - V(s)|\}$
      fin
    jusqu'à $\Delta \leq \epsilon$;
    pour tout $s \in S$ **faire**
      $b \leftarrow \Pi(s)$
      $\Pi(s) \leftarrow argmax_a\{\sum_{s' \in S} P(s'|s, a) \cdot V(s')\}$
      si $b \neq \Pi(s)$ **alors**
        $politique\_stable \leftarrow faux$
      fin
    fin
  jusqu'à $politique\_stable = vrai$;
  Retourner (Π)
fin

Algorithme 10 – L'algorithme de *Policy Iteration*

de l'algorithme consiste en deux étapes, l'évaluation et l'amélioration. La première étape nécessite au pire des cas $|S|^3$ opérations, tandis que la seconde s'effectue en $|A||S|^2$ opérations.

4.5.2 L'algorithme de *Value Iteration*

Dans l'algorithme *Policy Iteration*, chaque itération comprend l'évaluation d'une politique, ce qui exige plusieurs passages sur tous les états. Pouvoir interrompre la phase d'évaluation sans perdre la garantie de la convergence devient une opération importante. Une des méthodes permettant de réaliser cette rupture consiste à évaluer une fois pour toute la valeur $V(s), \forall s \in S$. Cette méthode, appelée *Value Iteration*, est une combinaison de la phase de l'amélioration et de la phase de l'évaluation rompue. L'itération dans cet algorithme est conservée pour la phase d'amélioration. L'algorithme *Value Iteration* est le suivant :

Données : Un MDP fini, un petit réel positif ϵ
Résultat : une politique optimale avec une marge d'erreur $< \epsilon$
début
  Initialisation quelconque de $V(s)$, par exemple $V(s) = 0, \forall s \in S$
  répéter
    $\Delta \leftarrow 0$
    pour tout $s \in S$ **faire**
      $v \leftarrow V(s)$
      $V(s) \leftarrow \alpha(s) + \gamma.\{\sum_{s' \in S} P(s'|s,a) \cdot V(s')\}$
      $\Delta \leftarrow max\{\Delta, |v - V(s)|\}$
    fin
  jusqu'à $\Delta \leq \epsilon$;
  Retourner $\Pi'(s) = argmax_a\{\sum_{s' \in S} P(s'|s,a) \cdot V(s')\}$
fin

Algorithme 11 – L'algorithme de *Value Iteration*

L'efficacité de l'algorithme *Value Iteration* dépend de deux facteurs, à savoir : la complexité d'une itération et le nombre d'itérations nécessaires pour converger. Chaque itération consiste à calculer la valeur de transition entre les états, pour chaque action : cela nécessite $|A||S|^2$ opérations.

4.5.3 Extensions de MDP

En MDPs, toute action est considérée comme ayant une durée d'une unité de temps, c.-à-d. qu'une action commence à t et se termine à $t + 1$. Une autre approche SMDPs[34], basée

[34] Semi-MDP.

sur les MDPs, permet une modélisation plus riche du temps et des actions [Garcia *et al.* 00, Hanna 03, Beynier 03, Le-Gloannec *et al.* 05, Beynier & Mouaddib 05, Beynier & Mouaddib 06] . En effet, dans les SMDPs, les actions peuvent avoir différentes durées, c.-à-d. qu'une action commence à t et se termine à $t + \Delta t$ où Δt est la durée de l'action.

Plusieurs autres types de MDPs ont été développés, nous citons POMDP[35] où l'état de l'environnement n'est pas toujours complètement observable, NOMDP[36] où l'état de l'environnement n'est pas du tout observable, FOMDP[37] où l'état de l'environnement est totalement observable. Pour plus de précisions, voir [Bellman 57].

Une des difficultés majeures du problème réside dans la taille du MDP qui est, en général, trop importante pour qu'il puisse être traité.

Nous n'utiliserons pas ce type de modèle dans notre application, car notre problème ne se prête pas bien à ce genre de modélisation de l'incertitude et il va poser l'exploit du MDP.

4.6 Limites de la planification en incertitude

Nous avons développé dans ce chapitre les planificateurs qui travaillent dans un environnement incertain conditionnel ou probabiliste, ou les deux à la fois. Nous avons aussi abordé les planificateurs qui combinent le temps et la probabilité dans un même système. Enfin nous avons introduit le processus de Markov. Ces planificateurs suppriment l'hypothèse de la certitude de l'environnement appliquée dans les planificateurs classiques et les planificateurs temporels. Certains planificateurs combinent dans un même système l'utilisation du temps et de l'incertitude en forme de probabilité. Le point faible de planificateurs existants de cette famille, est qu'ils sont applicables seulement à des problèmes bien déterminés, comme les opérations militaires ou les compagnies de voyages.

4.7 Notre approche

Cette première partie nous a permis de préciser le cadre de nos travaux. Nous avons vu que les planificateurs classiques font beaucoup d'hypothèses ce qui entraîne une limitation des problèmes pouvant être résolus par ce type de planificateurs. Les planificateurs temporels dépassent les planificateurs classiques en ajoutant les contraintes temporelles mais ils sont toujours limités à des problèmes bien spécifiques et ne traitant qu'une seule sorte de contraintes temporelles (quantitative ou qualitative). Les planificateurs en état d'incertitude ont dépassé les planificateurs classiques pour résoudre de problèmes plus réels. Les planificateurs temporels probabilistes

[35] Partailly-Observable MDP.
[36] Non-Observable MDP.
[37] Fully-Observable MDP.

combinent les contraintes temporelles et l'incertitude, mais la plupart de ceux-ci sont spécifique à un seul problème bien précis et ne peuvent pas être appliqués à d'autres problèmes. Le tableau 4.2 résume les caractéristiques principales de quelques planificateurs étudiés dans cette première partie.

Dans cette thèse, nous proposons un planificateur qui combine dans un même système le temps (quantitatif et qualitatif), le coût, l'incertitude (probabilité), l'utilisation du graphe ET/OU (plus général que le graphe OU et le graphe ET) et les préférences de l'utilisateur (problème multi-critères).

Nous proposons une approche de planification temporelle et probabiliste permettant aux tâches de respecter des contraintes temporelles ainsi que des contraintes de précédence. Chaque tâche possède un ensemble de contraintes temporelles, un ensemble de probabilités sur les durées d'exécution et un ensemble de coûts. Le planificateur ne peut travailler sans avoir une connaissance minimale de l'environnement dans lequel il évolue. Nous faisons l'hypothèse que notre planificateur dispose d'une base de données contenant toutes les durées possibles d'exécution, leurs coûts respectifs et qu'il connaît la distribution de probabilités associée à chaque durée.

contraintes / Planificateur	temps	incertitude	coût	graphe ET/OU
Planificateurs classiques	non	non	non	oui/non
Planificateurs temporels	oui	non	oui/non	non
Planificateurs en incertitude	non	oui	non	non
Planificateurs temporels probabilistes	oui	oui	oui/non	non
Notre planificateur	**oui**	**oui**	**oui**	**oui**

TAB. 4.1 – Notre planificateur prend en compte le temps, l'incertitude, le coût et utilise un graphe ET/OU. oui/non signifie qu'il y a au moins un planificateur de cette catégorie qui prend en compte la contrainte en question ; par exemple, parmi les planificateurs classiques cités dans cette thèse, seul le planificateur IPP utilise la méthode de recherche dans l'espace du graphe ET/OU

Une relation de précédence relie les tâches dans un graphe ET/OU. Grâce à une propagation temporelle à travers le graphe, nous calculons les intervalles d'exécution des tâches. Puis nous choisissons le meilleur plan pouvant être exécuté en respectant toutes les contraintes et selon les préférences de l'utilisateur. La figure 4.5 illustre les caractéristiques principales de notre planificateur.

Notre objectif est de réaliser un système général de planification et d'ordonnancement capable de prendre en compte un grand nombre de contraintes dans un environnement dynamique. Le tableau 4.1 représente les différents types de contraintes utilisés dans notre planificateur et dans

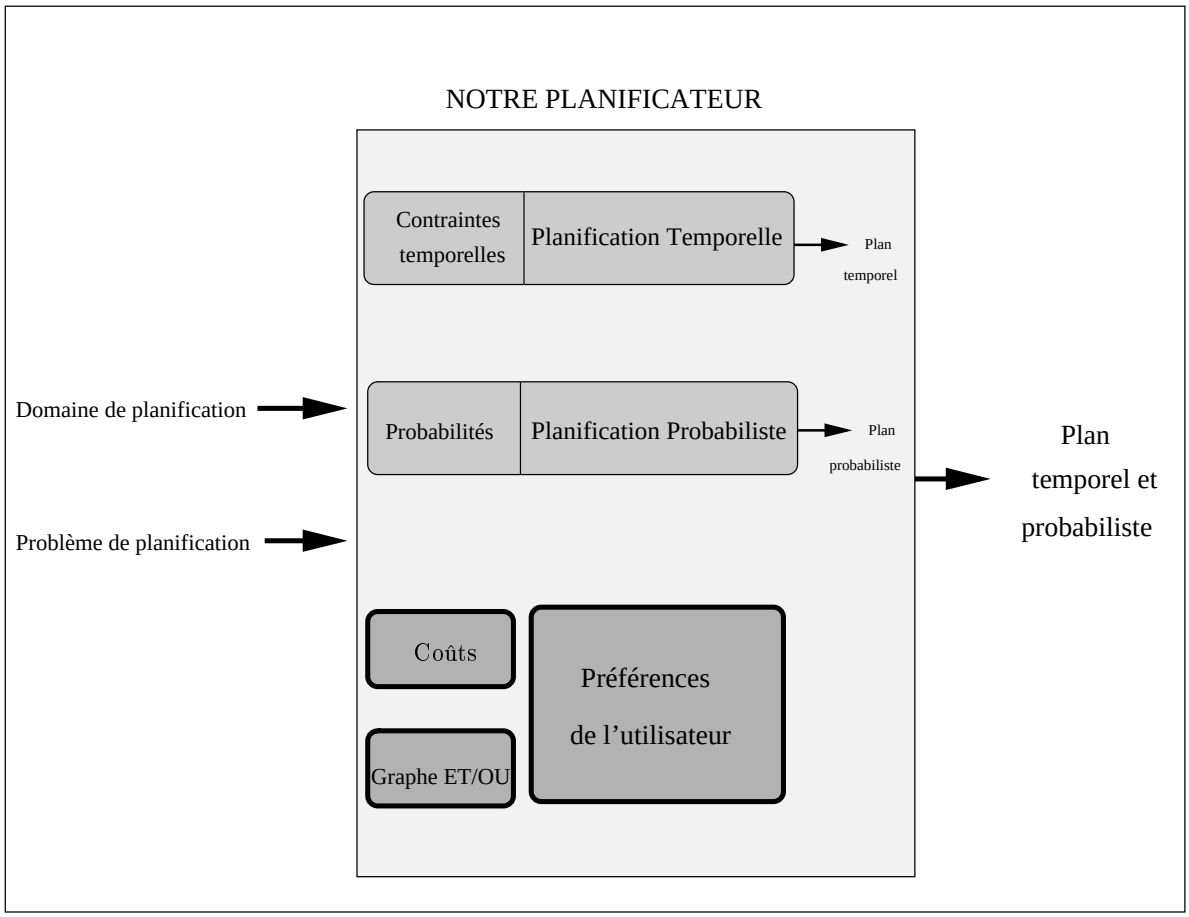

FIG. 4.5 – Notre planificateur

les autres.

Nous allons maintenant, dans la deuxième partie, le présenter de manière plus détaillée. Cette deuxième partie présente notre travail de thèse, c'est-à-dire la conception d'un nouveau système de planification.

Planificateurs classiques :	
STRIPS	environnement statique, observabilité totale, actions déterministes
GraphPlan	recherche dans un graphe de planification en chaînage arrière
IPP	recherche dans le graphe de planification en chaînage arrière graphe ET/OU
LCGP	recherche dans le graphe de planification en chaînage arrière
Alt-Alt	recherche dans le graphe de planification en utilisant la méthode de recherche dans l'espace d'états en chaînage arrière
UCPOP	recherche dans l'espace de plans et techniques de raffinement
ProPlan	recherche dans l'espace d'états
PL-Plan	recherche dans l'espace d'états
Planificateurs temporelles :	
Deviser	planification non-linéaire, fenêtres temporelles, durées
IxTeT	planification non-linéaire, hors ligne, recherche dans l'espace de plans partiels
CPT	recherche dans l'espace de plans partiels
TGP	recherche dans un graphe de planification en chaînage arrière
parcPlan	temps, coût, planification non-linéaire
Zeno	recherche dans l'espace de plans avec le moindre engagement
Sapa	recherche dans l'espace d'états en chaînage avant, coût
Planificateurs probabilistes :	
Buridan	recherche dans l'espace de plans partiels
PGraphPlan	recherche en arrière dans le graphe de planification
TGraphPlan	recherche en arrière dans le graphe de planification
MaxPlan	non observable
Probapop	Ordre partiel, non observable
Planificateurs conditionnels probabilistes :	
C-BURIDAN	recherche dans l'espace de plans partiels,observable, raffinements
Mahinur	recherche dans l'espace de plans partiels, raffinements
Planificateurs temporels probabilistes :	
Planificateur d'un voyage	recherche dans l'espace de plans, coût, temps
Planificateur des opérations militaires	recherche dans l'espace d'états, coût, temps
Prottle	recherche dans un graphe de planification ET/OU, coût, temps

TAB. 4.2 – Les méthodes de recherche utilisées dans les planificateurs cités dans cette thèse

Deuxième partie

Les contributions

Chapitre 5

Génération de plans

Un problème de planification temporelle est spécifié par l'environnement de la planification (un ensemble d'états, un ensemble d'actions, un état initial, un ensemble de buts et un ensemble de contraintes de précédence) et des caractéristiques temporelles (durées, dates de début, dates de fin, temps limites,...). La plupart des formalismes [Dechter *et al.* 91] permettent de représenter des contraintes sur les durées d'exécution des tâches. Cependant d'autres types de contraintes peuvent être envisagés. En particulier, dans les applications réelles, une incertitude sur les durées d'exécution des tâches est possible. Par exemple, dans un problème de transport, la durée d'un trajet n'est pas connue avec exactitude (cela dépend du climat, du trafic, etc). Il est donc important de pouvoir raisonner sur le temps et sous incertitude. Nous nous intéressons au problème de la planification et de l'ordonnancement sous contraintes temporelles et incertitude.

Notre problème est représenté par un graphe ET/OU où les nœuds représentent des tâches, qualifiées de contraintes de temps, de coût et de probabilités représentant l'incertitude sur la durée d'exécution des tâches. Les arcs quant à eux représentent les relations de précédence entre les tâches. Étant donné ce graphe, notre objectif est de déterminer un plan de tâches qui satisfait toutes les contraintes et qui a une grande probabilité d'être exécuté durant un temps et avec un coût réduits.

Autrement dit, étant donnés :

- la description du domaine (le graphe, les données temporelles, les coûts, les probabilités,...),
- un ensemble d'états initiaux,
- un ensemble de buts à atteindre, et
- un ensemble de contraintes sur les tâches, correspondant à un ordre partiel que l'on souhaite retrouver dans l'ordre de réalisation des buts,

notre planificateur permet de déterminer l'ensemble des plans possibles (appelés *plans faisables*) puis par affinements successifs, il en sélectionne un sous-ensemble appelé ensemble des *plans admissibles*. Parmi ces plans, il choisit, selon un ou plusieurs critères de préférence, le meilleur

plan à exécuter. La figure 5.1 représente les trois étapes principales de planification qui sont utilisées pour produire le meilleur plan à exécuter. Rappelons que nous ne nous intéressons pas à l'étape d'exécution des plans produits.

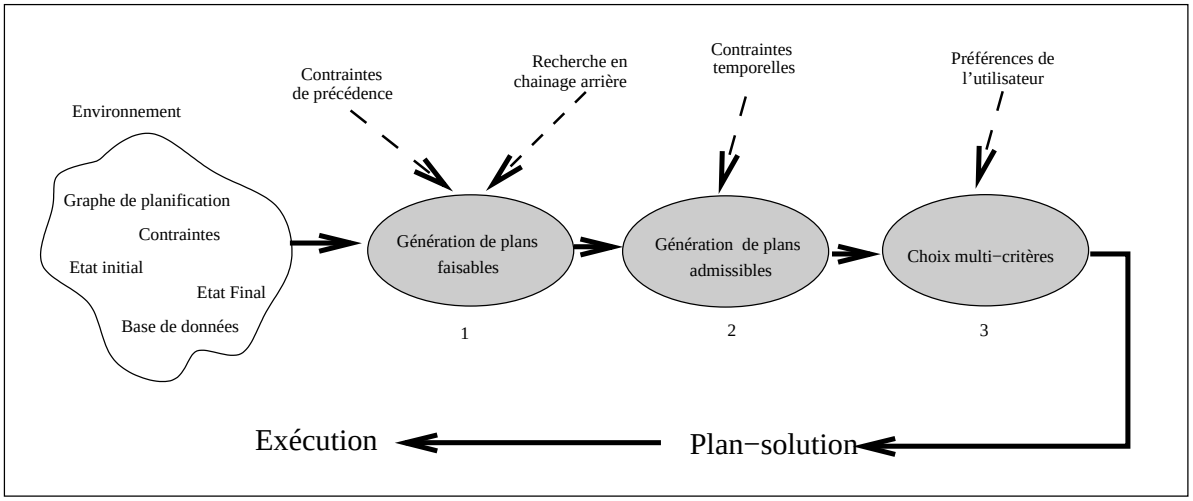

FIG. 5.1 – Les trois étapes principales de la planification

Une particularité importante du problème considéré est que les tâches n'ont pas une seule durée d'exécution, mais un ensemble de durées d'exécution pondérées par un ensemble de probabilités d'exécution. Cela exprime notamment des incertitudes sur la connaissance exacte de la durée d'exécution des tâches qui ne sera réellement connue qu'à l'exécution effective.

Ce chapitre présente la modélisation détaillée de notre problème. Nous y définissons l'environnement dans lequel le planificateur évolue, les méthodes de recherche utilisées et les deux premières étapes de génération de plans : génération des plans faisables et génération des plans admissibles.

5.1 Représentation des connaissances

Nous travaillons dans un système centralisé, c'est-à-dire que nous avons un seul agent qui a une vue globale sur l'environnement et qui dispose d'une base de connaissances contenant l'ensemble des informations concernant les tâches à exécuter. Cet agent central (le planificateur), représenté dans la figure 5.2, est responsable de la planification et de l'ordonnancement d'un ensemble de tâches dont l'exécution doit atteindre l'objectif désiré en respectant toutes les contraintes.

Avant d'aller plus loin, nous décrivons tout d'abord ce qu'est un agent et nous introduisons quelques définitions et notions que nous utiliserons dans la suite.

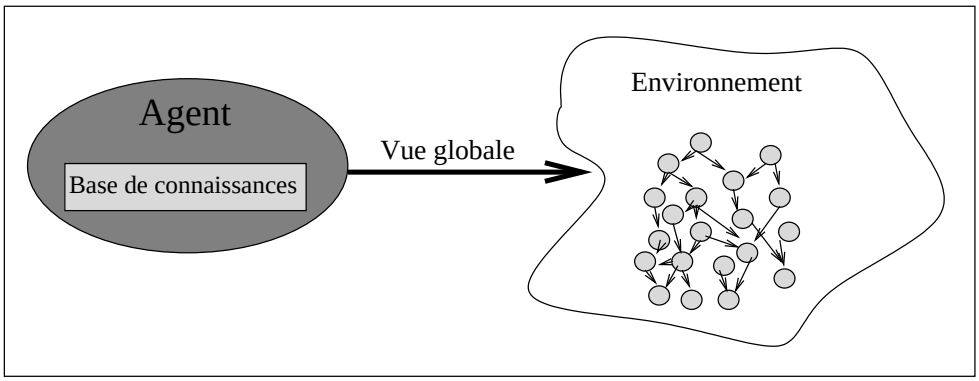

FIG. 5.2 – L'agent central a une vue globale sur l'environnement et dispose d'une base de données

5.1.1 Agent

Le terme *agent* est apparu dans la littérature informatique avec l'arrivée de programmes suffisamment sophistiqués en terme d'autonomie et d'intelligence pour que l'on puisse le considérer comme une entité différente du logiciel traditionnel. Plusieurs définitions ont été proposées pour décrire un agent et lui attribuer des propriétés. Wooldrige [WEISS 99] définit un agent comme un programme informatique évoluant dans un certain environnement et ayant la capacité d'agir de manière autonome afin d'atteindre des objectifs précis. Un agent selon Russel et Norvig [RUSSELL & NORVIG 03] est un système capable de décider par lui-même ce qu'il doit faire pour atteindre ses objectifs.

Malgré la diversité des définitions d'un agent, il y a accord sur le fait qu'un agent doit exercer un processus cognitif lui permettant d'analyser l'environnement et de déterminer les meilleurs actions à entreprendre.

C'est ce processus par lequel l'agent développe sa stratégie d'actions que nous appelons la *planification*. Le but de l'agent de planification est de choisir, gérer et ordonner un sous-ensemble de tâches à exécuter parmi un ensemble de tâches représentées dans un graphe ET/OU acyclique orienté.

5.1.2 Tâche

Une *tâche*, représentée dans la figure 5.3, est l'action réalisée par un agent pendant une durée du temps. À une tâche t est associée la liste $\langle I_t^-, I_t^+, \Delta_t, Pr_t, C_t \rangle$ où :

- I_t^- est la date à laquelle l'agent peut, au plus tôt, commencer l'exécution de t ;
- I_t^+ est la date à laquelle l'agent doit, au plus tard, avoir terminé l'exécution de t (échéance) ; L'intervalle $[I_t^-, I_t^+]$ est appelé la fenêtre temporelle d'exécution de t.
- $\Delta_t = \{d_t^1, d_t^2, ..., d_t^m\}$ est l'ensemble des durées d'exécution possibles de la tâche t ;

- $Pr_t = \{pr_t^1, pr_t^2, ..., pr_t^m\}$ est un ensemble de probabilités où pr_t^i est la probabilité que la tâche t s'exécute pendant la durée d_t^i. On a : $0 \leq pr_t^i \leq 1$ et $\sum_i^m pr_t^i = 1$;
- $C_t = \{c_t^1, c_t^2, ..., c_t^m\}$ est un ensemble de coûts, où c_t^i est le coût de l'exécution de la tâche t pour une durée d_t^i.

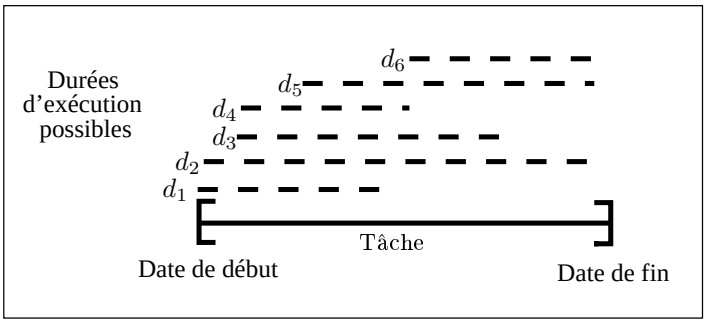

FIG. 5.3 – Une tâche possède une date de début au plus tôt, une date de fin au plus tard, un ensemble de durées d'exécution possibles ; à chacune de ces durées sont associés une probabilité et un coût d'exécution.

Nous décrivons dans ce qui suit, les modèles de temps, de probabilité et de coût des tâches que nous considérons dans notre recherche.

5.1.2.1 Modélisation du temps

Nous utilisons le temps sous la forme de (1) contraintes temporelles quantitatives (ou numériques) : une tâche doit être exécutée dans une fenêtre temporelle pendant une des durées d'exécution possibles qui sont représentées d'une façon linéaire et discrète et (2) contraintes temporelles qualitatives (ou symboliques) : les tâches sont reliées par des contraintes d'antériorité indiquant l'ordre d'exécution entre elles (tâche t_i avant ou après tâche t_j) (section 5.1.3).

Dans cette thèse, nous utilisons les termes "contraintes temporelles" pour désigner les contraintes temporelles quantitatives (fenêtres temporelles, durées, délais, ...) et "contraintes de précédence" pour désigner les contraintes temporelles qualitatives (d'antériorité).

5.1.2.2 Modélisation de l'incertitude

L'incertitude dans notre modèle est représentée sous la forme d'une distribution de probabilités sur l'ensemble des durées d'exécution possibles d'une tâche. À noter que cette probabilité ne signifie en aucun cas le degré de sucées d'exécution de la tâche. Nous verrons dans la suite de ce chapitre que la probabilité de l'exécution d'une tâche pendant une durée quelconque est une probabilité conditionnelle puisqu'elle dépend, en plus de sa probabilité d'exécution pendant cette durée, des probabilités des durées des tâches précédentes. La représentation de l'incertitude

par des probabilités sur les durées d'exécution est automatiquement étendue par la suite pour représenter les probabilités d'exécution des tâches et par conséquence celles des plans.

5.1.2.3 Modélisation du coût

Le coût représente ce que nous payons si une tâche est exécutée en une durée quelconque. En d'autres termes, l'exécution d'une tâche pendant une durée quelconque coûte une certaine quantité d'énergies, de ressources, d'argents, ou d'autres. Nous utilisons le mot "coût" pour désigner un ou plusieurs types possibles de coût. Comme la probabilité, le coût dépend des durées d'exécution possibles et non pas seulement de la tâche elle-même. En effet, une tâche t coûte c_t^i unités de coût si elle s'exécute en d_t^i unités de temps.

5.1.2.4 Validité des tâches

Le planificateur ne peut pas travailler sans avoir une connaissance minimale de l'environnement dans lequel il évolue. Nous faisons l'hypothèse que notre planificateur dispose d'une base de connaissances contenant toutes les informations concernant les tâches, à savoir, les dates de début et de fin possibles, les durées possibles d'exécution, leurs coûts respectifs et la distribution de probabilités associée aux durées d'exécution. Pour qu'une tâche soit valide, il faut que ses données associées respectent les règles suivantes :

- $I_t^- \geq 0$, $I_t^+ > 0$, $d_t^i \geq 0$,
- $I_t^- < I_t^+$,
- $d_t^i \leq I_t^+ - I_t^- \ \forall d_t^i \in \Delta_t$,
- $d_t^i \neq d_t^j \ \forall d_t^i, d_t^j \in \Delta_t$,
- $0 \leq pr_t^i \leq 1$ et
- $\sum_i pr_t^i = 1$.

5.1.3 Contraintes de précédence

Dans le monde réel, la possibilité d'exécuter une tâche peut dépendre de plusieurs conditions, comme le temps, les ressources disponibles et/ou l'exécution d'autres tâches. Dans ce dernier cas, on parle de contraintes de précédence exprimées sous la forme d'une relation d'ordre partiel sur un ensemble de tâches T. Nous considérons deux sortes de contraintes de précédence :

1. *Contraintes de précédence conjonctives* : Soient $t, t_1, t_2, \ldots, t_n$ des tâches. Une contrainte de précédence conjonctive entre t et l'ensemble de tâches $t_1, t_2, \ldots, t_n$, notée $[t_1, t_2, \ldots, t_n] \rightarrow t$, signifie que t ne peut s'exécuter que si $t_1, t_2, \ldots, t_n$ sont toutes exécutées. Nous désignons par la suite cette contrainte par ET.
2. *Contraintes de précédence disjonctives* : Soient $t, t_1, t_2, \ldots, t_n$ des tâches. Une contrainte de précédence disjonctive entre t et l'ensemble de tâches $t_1, t_2, \ldots, t_n$, notée $t_1|t_2|\ldots|t_n \rightarrow t$,

signifie que t ne peut s'exécuter que si au moins une tâche parmi $t_1, t_2, \ldots, t_n$ est exécutée. Nous désignons par la suite cette contrainte par OU.

Un cas particulier est lorsque t a un seul prédécesseur t_i, dans ce cas on parle de *contraintes de précédence simple* et on la note par $t_i \rightarrow t$.

La figure 5.4 représente ces trois sortes de contraintes de précédence possibles entre les tâches d'exécution.

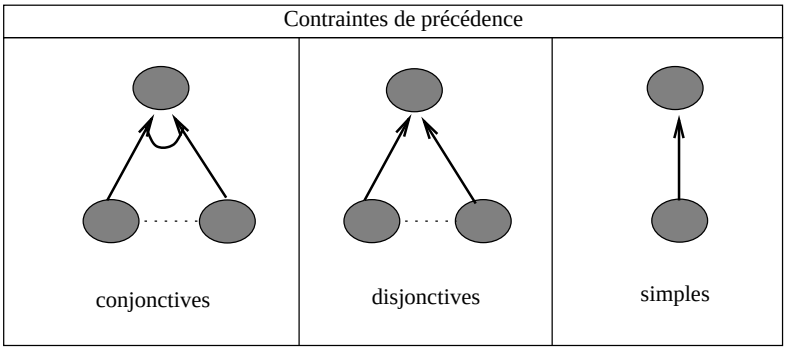

FIG. 5.4 – Les trois sortes de contraintes possibles entre les tâches d'exécution.

Nous ne traitons pas le cas où différents types de contraintes se réunissent en une même tâche (le cas où il faut avoir exécuté (t_1 ou t_2) et t_3 avant de commencer t_4). Cette modélisation de précédence est possible grâce à une simple transformation (ajout d'une tâche t_0 telle que $[t_1, t_3] \rightarrow t_0$, $[t_2, t_3] \rightarrow t_0$, $t_0 \rightarrow t_4$).

À noter que parfois on est amené à respecter un délai après la fin de l'exécution d'une tâche pour pouvoir exécuter la tâche suivante.

On désigne par δ_{t_i,t_j} le délai entre les tâches t_i et t_j et qui exprime que l'exécution de t_j ne peut commencer qu'après δ_{t_i,t_j} unités de temps de la fin de l'exécution de t_i.

5.1.4 Graphe ET/OU

Un graphe ET/OU associé à un agent est un graphe dont les nœuds représentent les tâches décrites dans la section 5.1.2 et les arcs représentent les relations de précédence de type ET et OU entre les tâches. Plus formellement :

Définition 1 *Soient T l'ensemble de tâches $t_1, t_2, \ldots, t_n$, E un ensemble de contraintes de précédence conjonctives et disjonctives tel que $E = \{c | c = [t_1, t_2, \ldots, t_m] \rightarrow t_i$ ou $c = t_1 | t_2 | \ldots | t_m \rightarrow t_i$ ou $c = t_m \rightarrow t_i$ où $t_1, t_2, \ldots, t_m, t_i \in T\}$ et D un ensemble de contraintes de délais tel que $D = \{\delta_{t_i,t_j} | \delta_{t_i,t_j} \in \mathbb{R}, t_i, t_j \in T$ et $\exists$ un arc de t_i à $t_j\}$, un graphe ET/OU est un graphe acyclique orienté noté $G = (T, E, D)$ où les nœuds sont des éléments de T et les arcs sont des éléments de E.*

Deux tâches dans le graphe ET/OU peuvent être reliées directement ($t_i \rightarrow t_j$) ou indirectement ($t_i \rightarrow t_j \rightarrow t_k$) par des contraintes de précédence conjonctives, disjonctives ou simples (l'exécution de l'une doit être absolument finie avant le début de l'exécution de l'autre), ou peuvent être indépendantes (leurs exécutions sont totalement indépendantes dans le temps).

Les tâches du graphe ET/OU ne sont pas toutes obligatoires. Cela est exprimé par l'existence de deux sortes de contraintes de précédence. Dans la solution du problème de planification que nous définirons plus tard dans ce chapitre, nous trouverons une ou plusieurs tâches à exécuter parmi l'ensemble de toutes les tâches formant le graphe ET/OU.

À noter que les nœuds qui forment l'ensemble T sont divisés en trois sous-ensembles. Nous écrirons $T = T_I \cup T_M \cup T_F$ où T_I est l'ensemble des tâches qui n'ont pas de prédécesseurs (les tâches initiales), T_M est l'ensemble des tâches qui ont des prédécesseurs et des successeurs et T_F est l'ensemble des tâches finales qui n'ont pas de successeurs.

5.1.5 Exemple Illustratif

Afin de bien présenter le problème abordé dans cette thèse et bien comprendre la façon de le résoudre, nous expliquons au fur et à mesure à l'aide d'un exemple toute notion introduite (un autre exemple est détaillé dans le chapitre 8).

Soit T un ensemble de tâches $t_1, t_2, \ldots, t_{18}$[38] reliées par des contraintes de précédence dans un graphe ET/OU et possédant chacune une liste de données (basée sur des études statistiques antérieures) à savoir une date de début, une date de fin, un ensemble de durées possibles, un ensemble de coûts associés aux durées et une distribution de probabilités sur l'ensemble de durées (section 5.1.2).

La figure 5.5 représente un graphe ET/OU $G = (T, E, D)$ tel que $T = \{t_1, \ldots, t_{18}\}$, E est l'ensemble des contraintes entre les tâches (représentées par des arcs) et D est l'ensemble de délais entre les tâches (représentés par des chiffres sur les arcs). Remarquons que $T = T_I \cup T_M \cup T_F$ tels que $T_I = \{t_1, t_2, t_3, t_4, t_5\}$, $T_F = \{t_{17}, t_{18}\}$ et $T_M = \{t_6, ..., t_{16}\}$.

Par exemple, la tâche t_2 a la liste des données suivante :$\langle [2,6], \{2,3\}, \{0.6, 0.4\}, \{10, 20\}\rangle$ c.-à-d. $I^-_{t_2} = 2$, $I^+_{t_2} = 6$, $d^1_{t_2} = 2$, $d^2_{t_2} = 3$, $pr^1_{t_2} = 0.6$, $pr^2_{t_2} = 0.4$, $c^1_{t_2} = 10$ et $c^2_{t_2} = 20$. Les tâches t_8, t_{10}, t_{11} et t_{17} sont des tâches ET, les tâches t_6 et t_{13} sont des tâches simples, et toutes les autres tâches sont des tâches OU. Plus précisément, l'exécution de t_{10} ne peut se faire qu'après la fin des exécutions de t_6 et de t_7, par contre l'exécution de t_7 peut se faire après la fin d'exécution de t_2 ou de t_3. L'exécution de t_6 ne peut se faire qu'après la fin d'exécution de t_1. Des délais d'attente existent entre les tâches, par exemple $\delta_{t_1,t_6} = 10$ signifie que l'exécution de t_6 ne peut se faire que 10 unités de temps après la fin de l'exécution de t_1.

Considérons un agent A, dont le but est de planifier et gérer l'ensemble de tâches T afin

[38] Les numéros attribués aux différentes tâches n'impliquent pas un ordre particulier d'exécution.

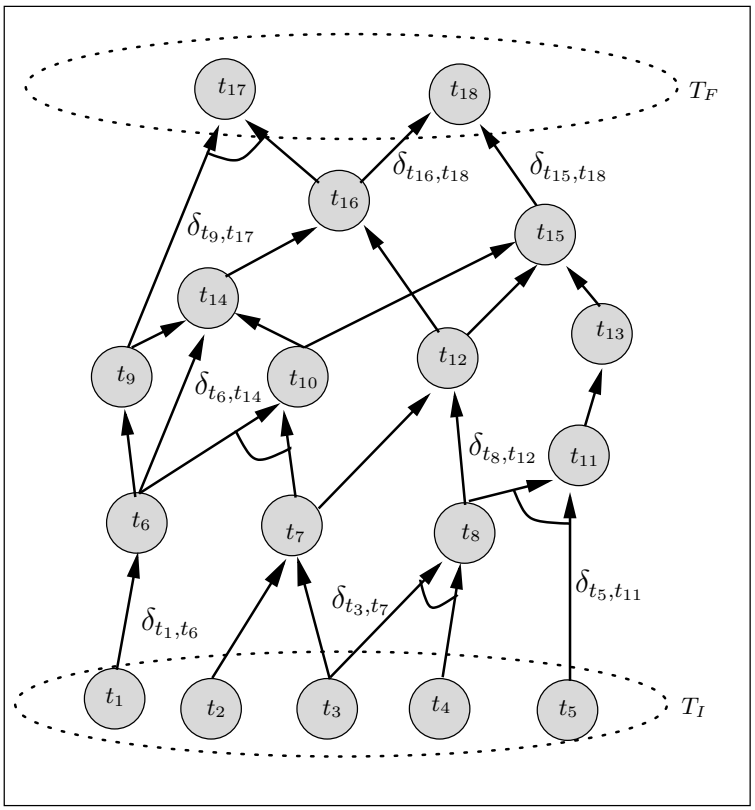

FIG. 5.5 – graphe ET/OU

d'assurer une exécution complète et efficace, et ayant une vue globale sur l'environnement. Étant donné un instant (*start_time*), l'agent doit déterminer le *"meilleur plan"* permettant d'arriver aux buts et respectant toutes les contraintes relatives aux tâches.

Le domaine de planification que nous considérons dans cette thèse est très contraint (relations de précédence entre les tâches, date de fin limitée, intervalles temporelles), concerne une situation à planifier dans un futur immédiat (l'exécution doit commencer tout de suite) et situé dans un environnement incertain (plusieurs durées probabilistes possibles pour chaque tâche). En fait, ce qui est important, c'est qu'il illustre une situation mettant en jeu des relations temporelles quantitatives et qualitatives, une distribution de probabilités et des coûts d'exécution. Le tout est formalisé en graphe ET/OU.

5.2 Génération de plans temporellement admissibles

Le problème que nous nous posons est, étant donné un graphe ET/OU décrit ci-dessus, de trouver le meilleur plan qui mène d'un ensemble de tâches initiales à un ensemble de buts de telle sorte que toutes les contraintes soient satisfaites. Un tel plan, appelé *plan-solution*, est un

sous-graphe du graphe allant d'un ensemble de tâches initiales T_I à un sous-ensemble de tâches finales $T' \subset T_F$. Nous définissons notre problème et sa solution comme suit :

Définition 2 *Un problème de planification est un couple (G, T'), où $G = (T, E, D)$ est un graphe ET/OU et T' est un sous-ensemble de l'ensemble T_F des tâches finales de G ($T' \subset T_F \subset T$).*

Définition 3 *Une solution au problème de planification est définie par un ensemble de tâches à exécuter de telle façon que tous les buts soient atteints et toutes les contraintes soient satisfaisantes.*

La recherche d'un plan-solution revient à comparer plusieurs plans alternatifs et à en choisir le meilleur. Pour résoudre le problème de génération de plan-solution, nous procédons en trois étapes principales (revoir la figure 5.1 dans l'introduction de ce chapitre) :

1. Génération des **plans faisables** : Un plan faisable est un sous-graphe du graphe ET/OU initial tel que les contraintes sont conjonctives ou simples. Ce plan permet d'arriver au but (sous ensemble des tâches finales) à partir d'un sous ensemble de tâches initiales. L'algorithme de recherche de plans faisables est basé sur un parcours de graphe en chaînage arrière et sur la méthode de recherche des plans partiels pour déterminer un plan. En d'autres termes, étant donné un plan incomplet contenant l'état final, l'objectif est d'arriver à un plan complet qui respecte toutes les contraintes de précédence.
2. Génération des **plans admissibles** : Un plan admissible est un plan faisable où toutes les contraintes temporelles de tâches et celles entre les tâches sont satisfaites. Dans cette étape, nous vérifions la validité de l'exécution possible des tâches des plans faisables en utilisant les fenêtres temporelles et les différentes durées d'exécution possibles des tâches. L'avantage de cette étape est qu'elle peut diminuer le nombre de tâches à traiter dans les étapes suivantes[39] puisque nous supprimons de l'ensemble des plans faisables celui qui contient une ou plusieurs tâches ne vérifiant pas les contraintes temporelles. Nous supprimons également de l'ensemble de durées d'exécution possibles de chaque tâche, la durée pendant laquelle il est impossible de l'exécuter. Pour cela, nous calculons tout d'abord les intervalles d'exécutions possibles de toutes les tâches des plans faisables. Ensuite, nous les validons en utilisant les fenêtres temporelles et les relations de précédence entre les tâches. Les plans considérés comme admissibles seront analysés selon les critères de coût, de temps et de probabilité.
3. Choix d'un **plan-solution**. Dans les deux étapes précédentes, nous nous sommes basés dans notre recherche sur les contraintes de précédence et les contraintes temporelles. Dans cette étape, nous utilisons les autres contraintes à savoir l'incertitude (exprimée sous la

[39] Au pire des cas, c.-à-d. si toutes les tâches des plans faisables sont valides, le nombre total des tâches à analyser par la suite est le même (les analyses et les tests sont détaillés dans le chapitre 9).

forme de probabilité) et le coût. Selon les préférences de l'utilisateur en terme de temps, de coût et de probabilité, l'objectif de cette étape est de sélectionner parmi tous les plans admissibles un seul plan à exécuter. Le compromis entre ces trois critères est une question vaste mais nous allons tâcher d'y répondre. L'utilisateur indique ses critères de préférence et le planificateur lui fournit le meilleur plan répondant à ses critères. Lorsque l'utilisateur préfère avoir un plan sûr d'être réalisé, le planificateur lui propose le plan le plus probable parmi les plans admissibles. Sinon, si l'utilisateur désire atteindre le but le plus tôt possible, le planificateur lui propose le plan le plus rapide. Selon le choix de l'utilisateur, le planificateur détermine (1) le plan le plus probable ayant un coût minimal, (2) le plan le plus probable s'exécutant pendant une durée minimale, (3) le plan le plus probable ayant un coût minimal et s'exécutant pendant une durée minimale, etc.

La sélection du meilleur plan selon un ou plusieurs critères fait l'objet du chapitre 6 de cette partie.

Ces trois étapes sont représentées sous la forme des ensembles dans la figure 5.6.

Dans ce qui suit, nous exposons de manière détaillée les deux étapes de la génération de plans à savoir la génération des plans faisables et celle des plans admissibles.

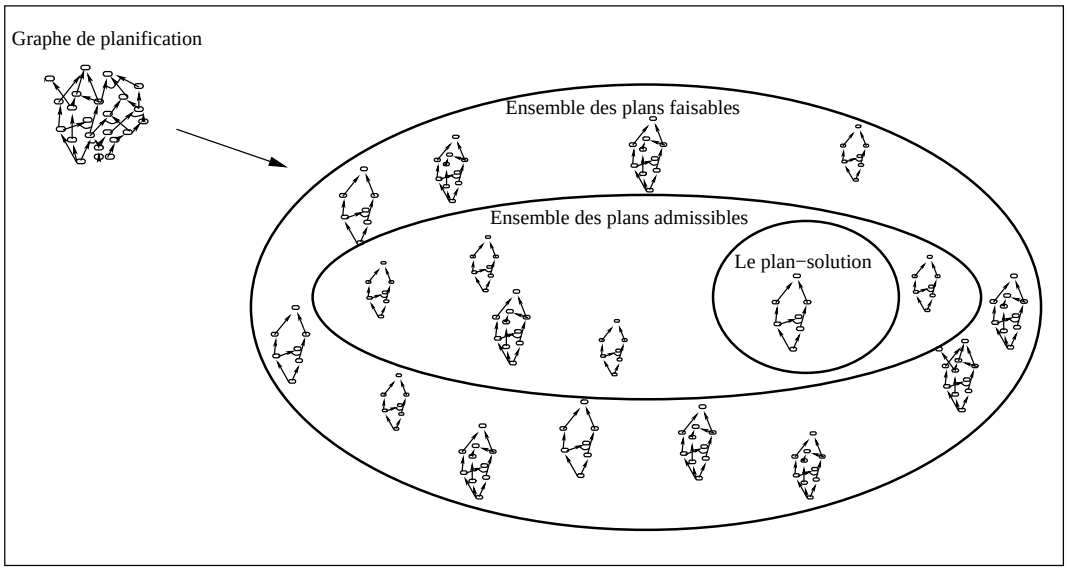

FIG. 5.6 – Les ensembles résultants des étapes de planification

5.2.1 Génération de plans faisables

Plusieurs méthodes de recherche ont montré leur efficacité dans la résolution de problèmes concernant la recherche des chemins dans un graphe. Ces méthodes sont en général fondées, comme expliqué dans la section 2.7 du chapitre 2, sur la recherche dans l'espace des états [FAR-

RENY 95] ou la recherche dans l'espace des plans [WELD 94, KAMBHAMPATI 97]. Chacune de ces deux méthodes utilise une des stratégies connues permettant de diriger une recherche : recherche en largeur, recherche en profondeur, *hill climbing*, recherche en avant, recherche en chaînage arrière, etc. Nous nous intéressons à trouver des solutions satisfaites plus qu'optimales. Nous cherchons les chemins dans le graphe ET/OU par un parcours de graphe en chaînage arrière avec la méthode de recherche des plans partiels. En d'autres termes, étant donné un plan incomplet contenant l'état final, l'objectif est d'arriver à un plan complet qui respecte toutes les contraintes de précédence. Autrement dit, le planificateur commence la recherche par la tâche finale ($t_{finale} \in T_F$) et s'arrête lorsqu'il trouve toutes les tâches initiales ($\in T_I$) permettant d'atteindre le but. Figure 5.7 illustre la première étape de planification : génération de plans faisables.

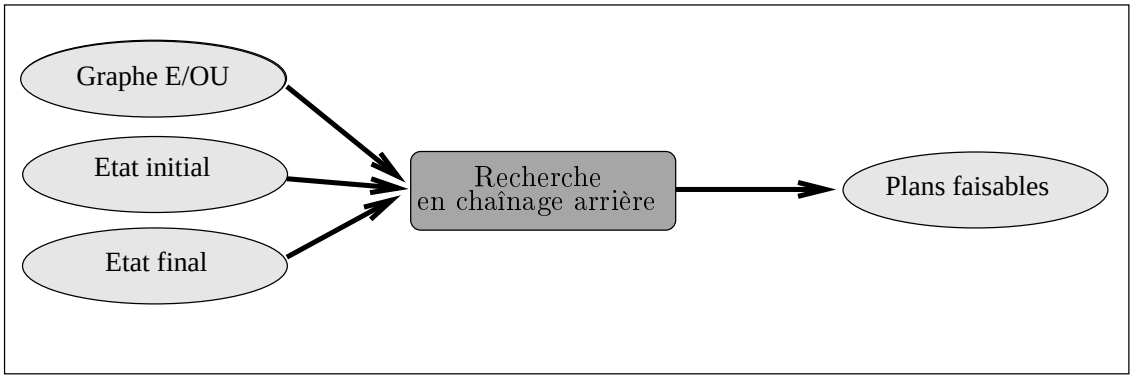

FIG. 5.7 – Génération de plans faisables

Exemple 1 *Les plans faisables de t_2 à t_{18} dans les graphe ET/OU de la figure 5.5, sont (figure 5.8) :*

- $[[t_2 \rightarrow t_7], [t_1 \rightarrow t_6]] \rightarrow t_{10} \rightarrow t_{14} \rightarrow t_{16} \rightarrow t_{18}$
- $[[t_2 \rightarrow t_7], [t_1 \rightarrow t_6]] \rightarrow t_{10} \rightarrow t_{15} \rightarrow t_{18}$
- $t_2 \rightarrow t_7 \rightarrow t_{12} \rightarrow t_{16} \rightarrow t_{18}$
- $t_2 \rightarrow t_7 \rightarrow t_{12} \rightarrow t_{15} \rightarrow t_{18}$

L'algorithme de recherche de l'ensemble des chemins amenant aux buts parcourt le graphe ET/OU en chaînage arrière et est intrinsèquement récursif (il s'appelle lui-même). Si ce processus est initié depuis la tâche finale t_{finale}, on finira par atteindre les tâches initiales par tous les chemins possibles. Cet algorithme est divisé en deux parties principales :

1. Recherche de chemins entre deux nœuds dans le graphe ET/OU. Tout d'abord, nous cherchons les chemins possibles entre chaque deux nœuds, en distinguant bien sûr les nœuds *OU* des nœuds *ET* ; si le chemin passe par un nœud *ET*, il faut que tous les prédécesseurs

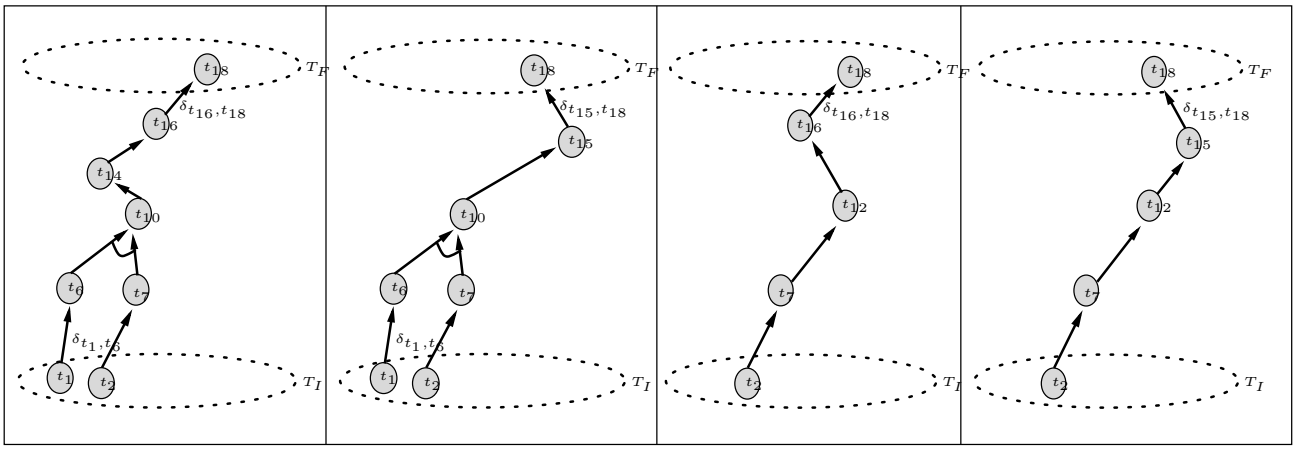

FIG. 5.8 – Les plans faisables sélectionnés de l'exemple 5.1.5

appartiennent déjà au chemin ; par contre lorsque le chemin passe par un nœud OU, nous regardons parmi ses prédécesseurs s'il existe un nœud qui appartient aussi au chemin. Si c'est le cas, nous ne conservons que celui-ci car d'après la définition d'un nœud *OU*, si un prédécesseur appartient au chemin, la tâche du nœud successeur peut être exécutée.

2. Reconstruction du sous-graphe dans le graphe initial. Nous utilisons les résultats obtenus dans la première étape de recherche et nous cherchons les chemins commençant par un nœud initial et terminant par nœud final en utilisant la méthode de recherche en arrière dans le graphe ET/OU. L'algorithme s'arrête quand nous trouvons le nœud initial cherché. Si aucun chemin trouvé, l'algorithme rend le résultat $NULL$.

Les deux parties de l'algorithme de recherche des plans faisables sont présentées dans les algorithmes 12 (la procédure "rechercheChemins" de cet algorithme est représenté dans l'algorithme 13) et 14 (la fonction "reconstruireGraphe" de cet algorithme est représenté dans l'algorithme 15).

5.2.2 Génération de plans admissibles

Après avoir calculé l'ensemble des plans faisables relatifs à un problème de planification (G, T') où G est un graphe ET/OU et T' est un sous-ensemble de l'ensemble des tâches finales, nous déterminons maintenant les plans **admissibles**. Pour se faire, nous calculons pour chaque plan faisable l'ensemble des intervalles d'exécution possibles de chacune de ses tâches. Ensuite, nous vérifions la possibilité d'exécuter les tâches pendant les intervalles d'exécution trouvés. Rappelons qu'un plan faisable est un sous-graphe où les nœuds sont reliés uniquement par des contraintes simples ou conjonctives. La figure 5.9 illustre la deuxième étape de planification : génération de plans admissibles.

Nous utilisons les techniques de propagation de contraintes temporelles inspirées de [BRESINA

Données :
- G : un graphe ET/OU
- *from* : le nœud de G d'où doit partir le chemin (le but)
- *to* : le nœud de G où doit aboutir le chemin (les tâches initiales)

Résultat :
- *tousLesSousGraphes* : la liste des sous-graphes minimaux engendrés par les chemins de *tousLesChemins*

début

tousLesChemins ← liste vide
chemin ←liste vide
rechercheChemins(*from, to, chemin, tousLesChemins*)
retourner *tousLesChemins*

fin

Algorithme 12 – Recherche de chemins entre deux nœuds dans le graphe ET/OU

Procédure rechercheChemins

Données :
- *courant* : un nœud de type ET/OU
- *destination* : le nœud où doit aboutir le chemin
- *chemin* : le chemin exploré jusqu'à présent
- *tousLesChemins* : la liste des chemins recherchés

Résultat :
- calcul de tous les chemins possibles entre *courant* et *destination*

début

ajouter *courant* à *chemin*
si *destination=courant* **alors**
  ajouter *chemin* à *tousLesChemins* et s'arrêter
fin
pour tout *fils de prédécesseur(courant)* **faire**
  rechercheChemins(*fils, destination, chemin, tousLesChemins*)
fin
retirer le dernier élément de *chemin*

fin

Algorithme 13 – Procédure rechercheChemins de l'algorithme 12

Données :
- G : un graphe ET/OU
- *tousLesChemins* : une liste de chemins contenus dans G

Résultat :
- *tousLesSousGraphes* : la liste des sous-graphes minimaux engendrés par les chemins de *tousLesChemins*

début

tousLesChemins ← nouvelle file d'attente remplie par les éléments de *tousLesChemins*

listeResultats ← liste vide

tant que *tousLesSousGraphes est non vide* **faire**

chemin ← premier élément de la file (pop)

racine ← reconstruireGraphe(*racine, chemin, tousLesSousGraphes*)

ajouter *racine* à *listeResultats*

fin

retourner *listeResultats*

fin

Algorithme 14 – Reconstruction des sous-graphes (chemins) dans le graphe initial

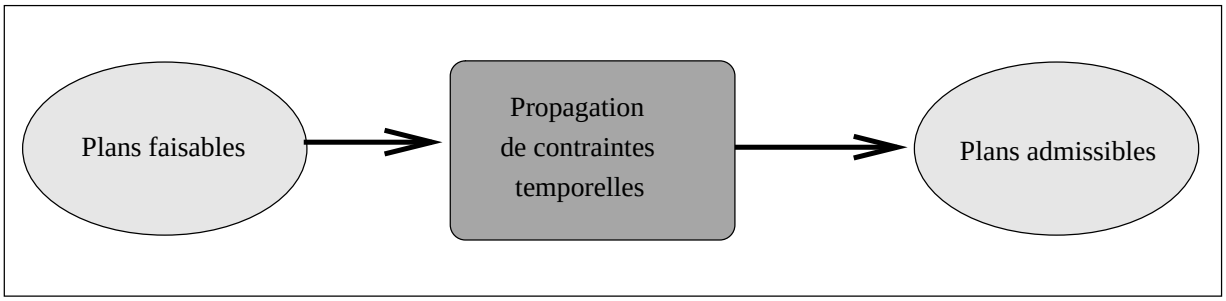

FIG. 5.9 – Génération de plans admissibles

& WASHINGTON 00] dans les plans faisables du graphe ET/OU qui a pour but de simplifier la résolution d'un problème. Après la recherche de tous les plans possibles dans le graphe (plans faisables), des affinements successifs éliminent des tâches voire des plans non admissibles à cause de la non-validité des contraintes temporelles quantitatives (par exemple, l'impossibilité d'exécuter une tâche pendant une durée possible quelconque si elle dépasse sa date de fin). Ce filtrage évite de nombreuses tentatives de résolution vouées à l'échec.

Définition 4 *On appelle intervalle d'exécution d'une tâche t tout intervalle contenu dans la fenêtre d'exécution* $[I_t^-, I_t^+]$ *dont la largeur appartient à l'ensemble* Δ_t *des durées d'exécution possibles.*

Fonction reconstruireGraphe

Données :
- *graphe* : le graphe de départ (où l'on cherche les chemins)
- *courant* : le nœud courant
- *file* : la file d'attente des graphes encore à reconstruire

Résultat :
- *nœudReconstruit* : le nœud à la racine du graphe engendré par le nœud courant et respectant les contraintes du graphe initial

début

résultat ← nouveau nœud
si *courant est un nœud ET* **alors**
pour tout *fils de prédécesseur(courant)* **faire**
n
fin
← reconstruireGraphe(*graphe, fils*)
ajouter *n* aux prédécesseurs de résultat
fin
si *courant est un nœud OU* **alors**
cheminContientUnFils ← FAUX
pour tout *fils de prédécesseur(courant)* **faire**
si *chemin contient fils* **alors**
cheminContientUnFils ← VRAI
fin
fin
si *(cheminContientUnFils = VRAI) ou (nombre de prédécesseurs de courant=1)*
alors
n1 ← reconstruireGraphe(*fils, graph*)
ajouter *n1* aux prédécesseurs de résultat
sinon
pour tout *fils1 de prédécesseur(courant)* **faire**
g1 = copie du *graphe*
newCourant ← l'équivalent de courant dans *g1*
retirer tous les prédécesseurs de *newCourant*, sauf l'équivalent de *fils1*
ajouter *g1* à la fin de la file d'attente
retourner *NULL*
fin
fin
fin
retourner *résultat*
fin

Algorithme 15 – La fonction reconstruireGraphe de l'algorithme 14

Soit la tâche t, nous désignons par s_t (respectivement e_t) une date de début d'exécution possible de t (respectivement une date de fin d'exécution possible de t), par S_t (respectivement E_t) l'ensemble de dates de début d'exécution possibles de t (respectivement l'ensemble de dates de fin d'exécution possibles de t) et par $\mathcal{I}$ l'ensemble des intervalles d'exécution possibles durant lesquels t peut s'exécuter. La différence entre une fenêtre temporelle et un intervalle d'exécution est qu'une fenêtre temporelle d'une tâche t représente la distance entre sa date de début au plus tôt I_t^- et sa date de début au plus tard I_t^+ et dont les durées d'exécution possibles sont inférieures ou égales à cette distance, tandis qu'un intervalle d'exécution de t commence par une date de début possible $s_t \geq I_t^-$ et se termine par une date de fin possible $e_t \leq I_t^+$ et s'exécute pendant une durée bien déterminée $d_t^i \in \Delta_t$. Une fenêtre temporelle est un intervalle d'exécution si la durée d'exécution possible $d_t^i \in \Delta_t$ est égale à $I_t^+ - I_t^-$. Tous les intervalles d'exécution doivent être à l'intérieure de la fenêtre temporelle de la tâche correspondante c.-à-d. que dans tous les cas, une fenêtre temporelle contient un intervalle d'exécution. La figure 5.10, représente les quatre

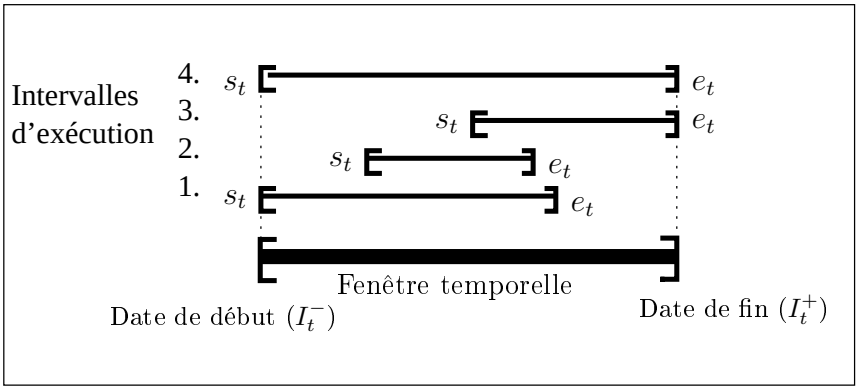

FIG. 5.10 – Possibilités de position d'un intervalle d'exécution par rapport à une fenêtre temporelle

possibilités de position d'un intervalle d'exécution $[s_t, e_t]$ par rapport à la fenêtre temporelle $[I_t^-, I_t^+]$ d'une tâche t :

1. $s_t = I_t^-$ et $e_t < I_t^+$,
2. $s_t > I_t^-$ et $e_t < I_t^+$,
3. $s_t > I_t^-$ et $e_t = I_t^+$,
4. $s_t = I_t^-$ et $e_t = I_t^+$ $(I_t^+ - I_t^- = d_t^i)$.

L'exécution des tâches peut se faire en parallèle (en partie ou au total) ou en séquentiel. Cela dépend des contraintes temporelles qualitatives (précédence) et aussi de la position des tâches dans le graphe. Deux tâches successeurs dans le graphe doivent absolument s'exécuter l'une après l'autre même si la fenêtre temporelle de l'une contient, intercale ou chevauche celle de l'autre.

Dans ce qui suit, nous décrivons la méthode de calcul des intervalles d'exécution pour chaque tâche d'un plan faisable. Ceux-ci sont calculés avant le début d'exécution effective des tâches. Nous utilisons un algorithme basé sur une propagation temporelle dans le graphe ET/OU [Bresina & Washington 00]. Cette propagation organise le graphe en plusieurs niveaux : l_0 contient les tâches initiales du graphe, l_1 contient toutes les tâches qui succèdent directement aux tâches du niveau l_0 et l_i contient toutes les tâches qui succèdent directement aux tâches du niveau l_{i-1}. Les feuilles sont les nœuds qui n'ont pas de successeurs. Pour chaque tâche d'un niveau donné l_i, nous calculons tous les intervalles d'exécution possibles en tenant compte des contraintes temporelles de la tâche et des contraintes temporelles de ses prédécesseurs.

5.2.3 Calcul des intervalles d'exécution des tâches dans un plan faisable

Soit une tâche t appartenant à l'ensemble des tâches d'un plan faisable $\mathcal{P}_f$ ($t \in T(\mathcal{P}_f)$). Soient $\Delta_t = \{d_t^1, d_t^2, ..., d_t^m\}$ l'ensemble des durées d'exécution de la tâche t, I_t^- la date à laquelle l'agent peut au plus tôt commencer l'exécution de t et I_t^+ la date à laquelle l'agent doit au plus tard terminer l'exécution de t. Supposons que l'agent peut commencer l'exécution à partir d'une date donnée appelée $start\_time$[40].

5.2.3.1 Cas d'une tâche initiale

Les dates de début et de fin d'exécution possibles d'une tâche initiale t appartenant au niveau l_0 sont calculées comme suit :

1. $S_t = \{max(I_t^-, start\_time)\}$, l'agent peut commencer l'exécution de la tâche à partir du moment où on le lui demande. Si $start\_time$ est plus grande que la date de début au plus tôt de la fenêtre temporelle de la tâche, un temps d'attente de I_t^- à $start\_time$ est alors nécessaire et donc l'exécution ne commence qu'à $start\_time$.
2. Pour chaque durée $d_t^i \in \Delta_t$ ($i = 1..m$, m est le nombre de durées d'exécution de t), l'agent calcule la date de fin possible s'il commence l'exécution de la tâche à la date du début déterminée ci-dessus. L'ensemble de fins d'exécution possibles est $E_t = \{e_t^i$ tel que $e_t^i = s_t + d_t^i\}$.
3. Les intervalles d'exécution possibles de t sont composés d'une date de début d'exécution possible et d'une date de fin d'exécution possible, $\mathcal{I}_t = \{I_t^i = [s_t, e_t^i]$ où $i = 1..m\}$, I_t^i est un intervalle d'exécution possible de t.

Pour chaque tâche initiale, nous obtenons une seule date de début d'exécution possible et plusieurs dates de fin d'exécution possibles. Le nombre d'intervalles d'exécution d'une tâche initiale

[40] $start\_time$ est la date de début qu'on indique à l'agent pour lui dire qu'à partir de cette date il peut commencer la planification et ensuite l'exécution.

t est égal au nombre des durées $d_t^i \in \Delta_t$ qui satisfont la condition :

$$max(start\_time, I_t^-) + d_t^i \leq I_t^+$$

où $[I_t^-, I_t^+]$ est la fenêtre temporelle de la tâche t.

Exemple 2 *Rappelons qu'à une une tâche t est associée la liste* $\langle [I_t^-, I_t^+], \Delta_t, Pr_t, C_t \rangle$.
Supposons que les tâches du plan faisable $[[t_2 \rightarrow t_7], [t_1 \rightarrow t_6]] \rightarrow t_{10} \rightarrow t_{15} \rightarrow t_{18}$ *ont les données suivantes :*

t_1 : $\langle [0,6], \{5,6\}, \{0.2, 0.8\}, \{12, 20\} \rangle$
t_2 : $\langle [2,6], \{2,3\}, \{0.6, 0.4\}, \{10, 20\} \rangle$
t_6 : $\langle [6,16], \{8,9,10\}, \{0.3, 0.3, 0.4\}, \{8, 10, 20\} \rangle$
t_7 : $\langle [5,22], \{15,17\}, \{0.5, 0.5\}, \{40, 45\} \rangle$
t_{10} : $\langle [20,30], \{4,6,10\}, \{0.1, 0.3, 0.6\}, \{2, 8, 20\} \rangle$
t_{15} : $\langle [26,36], \{5,10\}, \{0.4, 0.6\}, \{12, 22\} \rangle$
t_{18} : $\langle [30,50], \{14,15,20\}, \{0.2, 0.6, 0.2\}, \{40, 50, 70\} \rangle$
Supposons que $start\_time = 1$, *les intervalles d'exécution possibles de* t_1 *et de* t_2 *sont calculés comme suit :*

$s_{t_1} = max(I_{t_1}^-, start\_time) = max(0,1) = 1$ $e_{t_1}^1 = s_{t_1} + d_{t_1}^1 = 1 + 5 = 6$ $e_{t_1}^2 = s_{t_1} + d_{t_1}^2 = 1 + 6 = 7$ $S_{t_1} = \{1\}$ et $E_t = \{6, 7\}$	$s_{t_2} = max(I_{t_2}^-, start\_time) = max(2,1) = 2$ $e_{t_2}^1 = s_{t_2} + d_{t_2}^1 = 2 + 2 = 4$ $e_{t_2}^2 = s_{t_2} + d_{t_2}^2 = 2 + 3 = 5$ $S_{t_2} = \{2\}$ et $E_t = \{4, 5\}$
$\mathcal{I}_{t_1} = \{[1,6], [1,7]\}$	$\mathcal{I}_{t_2} = \{[2,4], [2,5]\}$

Remarquons que l'intervalle $I_{t_1}^2 = [1,7]$ *a une date de fin supérieure à* $I_{t_1}^+ = 6$. *Cet intervalle n'est donc pas valide. Si par contre nous avions* $I_{t_1}^+ = 5$, *alors tous les intervalles d'exécution de* t_1 *dans ce plan faisable ne seraient pas valides puisqu'ils dépassèrent* $I_{t_1}^+$. *Par conséquent, nous considérons que la tâche* t_1 *ainsi que le plan qui la contient ne sont pas valides.*

5.2.3.2 Cas d'une tâche intermédiaire ou finale

Pour $i \neq 0$, les dates de début d'exécution possibles de chaque tâche t appartenant au niveau l_i, sont calculées à partir des dates de fin des prédécesseurs de la tâche et de sa propre date de début au plus tôt possible (I_t^-). Rappelons qu'un plan faisable contient seulement de contraintes de précédence conjonctives et simples. La méthode de calcul n'est pas exactement la même pour les nœuds *OU* (contraintes simples) et pour les nœuds *ET* (contraintes conjonctives) du graphe ET/OU. En effet, pour qu'une tâche de type *OU* soit exécutée, il faut que la tâche prédécesseur soit exécutée. Pour qu'une tâche de type *ET* soit exécutée, il faut absolument que toutes les

tâches prédécesseurs soient exécutées.

Cas d'une tâche *OU*

Soit $t \in T_M \cup T_F$. On note toujours $\Delta_t = \{d_t^1, d_t^2, ..., d_t^m\}$ l'ensemble des durées d'exécution de la tâche t. Si t a un seul prédécesseur t' ($t' \rightarrow t$), avec $E_{t'} = \{e_{t'}^1, ..., e_{t'}^p\}$ alors les dates de début et de fin de t sont calculées de la manière suivante :

1. Déterminer l'ensemble S_t (initialisé à $\emptyset$) des dates de début d'exécution de t comme suit :
   - Pour $i = 1$ jusqu'à $i = p$ (p est le nombre de dates de fin possibles de t')
     $S_t = S_t \cup \{max(I_t^-, e_{t'}^i + \delta_{t',t})\}$.

   Chaque date de fin de la tâche t' précédant la tâche t est candidate pour devenir une date de début possible de t. Pour calculer les dates de début possibles de t, nous choisissons le maximum entre la date de début au plus tôt de sa fenêtre temporelle et chaque date de fin possible de t', bien sûr en tenant compte des délais d'attente. Chaque résultat obtenu représente une date de début possible pour t.
2. Déterminer l'ensemble E_t (initialisé à $\emptyset$) des dates de fin d'exécution de t comme suit :
   - Pour $i = 1$ jusqu'à $i = p$
     - Pour $j = 1$ jusqu'à $j = m$
       $E_t = E_t \cup \{e_t^{ij} = s_t^i + d_t^j\}$

   Les dates de fin d'exécution possibles de t sont calculées en ajoutant à chaque date de début trouvée à l'étape 1, ses différentes durées d'exécution possibles.
3. $I_t^{ij} = [s_t^i, e_t^{ij}]$ est un intervalle d'exécution possible de t tel que $s_t^i \in S_t$ et $e_t^{ij} \in E_t$.

$s_{t_6}^1 = max(I_{t_6}^-, e_{t_1}^1) = max(6,6) = 6$ (rappelons que $e_{t_1}^2 = 7$ n'est pas valide) $S_{t_6} = \{6\}$ $e_{t_6}^1 = s_{t_6}^1 + d_{t_6}^1 = 6 + 8 = 14$ $e_{t_6}^2 = s_{t_6}^1 + d_{t_6}^2 = 6 + 9 = 15$ $e_{t_6}^2 = s_{t_6}^1 + d_{t_6}^2 = 6 + 10 = 16$ $E_{t_6} = \{14, 15, 16\}$	$s_{t_7}^1 = max(I_{t_7}^-, e_{t_2}^1) = max(5,4) = 5$ $s_{t_7}^2 = max(I_{t_7}^-, e_{t_2}^2) = max(5,5) = 5$ $S_{t_7} = \{5\}$ $e_{t_7}^1 = s_{t_7}^1 + d_{t_7}^1 = 5 + 15 = 20$ $e_{t_7}^2 = s_{t_7}^1 + d_{t_7}^2 = 5 + 17 = 22$ $E_{t_7} = \{20, 22\}$
$\mathcal{I}_{t_6} = \{[6,14], [6,15], [6,16]\}$	$\mathcal{I}_{t_7} = \{[5,20], [5,22]\}$

Exemple 3 *Supposons, pour simplifier, que les délais entre les tâches sont égaux à zéro, le calcul des intervalles d'exécution de t_6 et de t_7 est représenté dans la table 3.*

De manière plus formelle, l'algorithme 16 résume le calcul des intervalles d'exécution possibles d'une tâche *OU*.

début
  $\mathcal{I}_t = \emptyset$
  pour tout $i = 1..p$ **faire**
    si $I_t^- > e_{t'}^i + \delta_{t',t}$ **alors**
      $s_t^i = I_t^-$
    sinon
      $s_t^i = e_{t'}^i + \delta_{t',t}$
    fin
    pour tout $j = 1..m$ **faire**
      $e_t^{ij} = s_t^i + d_t^j$
      $I_t^{ij} = [s_t^i, e_t^{ij}]$
      $\mathcal{I}_t = \mathcal{I}_t \cup I_t^{ij}$
    fin
  fin
fin

Algorithme 16 – Calcul des intervalles d'exécution possibles d'une tâche *OU*

Cas d'une tâche *ET*

Soient $t \in T_M \cup T_F$ et $\Delta_t = \{d_t^1, d_t^2, ..., d_t^m\}$ l'ensemble des durées d'exécution de la tâche t. Si t a un ensemble de prédécesseurs directs $t_1, t_2, \ldots, t_n$ ($[t_1, t_2, \ldots, t_n] \rightarrow t$) et si pour toute tâche t_i ($i = 1..n$), $E_{t_i} = \{e_{t_i}^1, e_{t_i}^2, ..., e_{t_i}^{j_i}\}$ alors les dates de début et de fin de t sont calculées de la manière suivante :

1. Déterminer l'ensemble S_t (initialisé à $\emptyset$) des dates de début d'exécution de t comme suit :
   - Calculer le maximum des dates de fin ($max(e_{t_i}^k + \delta_{t_i,t})$) des tâches précédentes de t.
   - $S_t = S_t \cup \{max(I_t^-, max(e_{t_i}^k + \delta_{t_i,t}))\}$

   En effet, chaque date de fin de chaque tâche $t_1, t_2, \ldots, t_n$ prédécesseur de la tâche t est candidate pour devenir une date de début possible de t. Une date de fin possible d'une tâche prédécesseur t_i est une date de début possible pour t si, après l'ajout du délai d'attente, elle est supérieure à la date de début au plus tôt de t et elle est supérieure à toutes les dates de fin possibles des autres tâches prédécesseurs de t.

2. Déterminer l'ensemble E_t (initialisé à $\emptyset$) des dates de fin d'exécution de t comme suit :
   - Pour $k = 1$ jusqu'à $k = p$ (p représente la cardinalité de l'ensemble S_t déterminé à l'étape 1)
     - Pour $r = 1$ jusqu'à $r = m$ (m est le nombre de durées d'exécution de t)

       $E_t = E_t \cup \{e_t^{kr} = s_t^k + d_t^r\}$

Les dates de fin d'exécution possibles de t sont calculées en ajoutant à chaque date de début trouvée à l'étape 1, ses différentes durées d'exécution possibles.

3. $I_t^{kr} = [s_t^k, e_t^{kr}]$ est un intervalle d'exécution possible de t tel que $s_t^k \in S_t$ et $e_t^{kr} \in E_t$.

De manière plus formelle, l'algorithme 17 calcule les intervalles d'exécution possibles d'une tâche ET.

début
  $max \leftarrow 0, \mathcal{I}_t = \emptyset$
  pour tout $i = 1..n$ **faire**
    pour tout $k = 1..j_i$ **faire**
      si $e_{t_i}^k + \delta_{t_i,t} > max$ **alors** $max = e_{t_i}^k + \delta_{t_i,t}$
    fin
  fin
  si $I_t^- > max$ **alors**
    $s_t^k = I_t^-$
  sinon
    $s_t^k = max$
  fin
  pour tout $r = 1..m$ **faire**
    $e_t^{kr} = s_t^k + d_t^r$
    $I_t^{kr} = [s_t^k, e_t^{kr}]$
    $\mathcal{I}_t = \mathcal{I}_t \cup I_t^{kr}$
  fin
fin

Algorithme 17 – Calcul des intervalles d'exécution possibles d'une tâche ET

Exemple 4 *Les intervalles d'exécution de* t_{10} *sont calculés dans la table 5.1. Remarquons que l'intervalle* $I_{t_{10}}^{23} = [22, 32]$ *a une date de fin supérieure à* $I_{t_{10}}^+ = 30$. *Cet intervalle n'est donc pas valide. Si par contre nous avions* $I_{t_{10}}^+ < 24$, *alors tous les intervalles d'exécution de* t_{10} *dans ce plan faisable ne seraient pas valides puisqu'ils dépassèrent* $I_{t_{10}}^+$. *Par conséquent, nous considérons que la tâche* t_{10} *ainsi que le plan qui la contient ne sont pas valides.*

Le nombre d'intervalles d'exécution de t dépend du nombre de dates de début d'exécution possibles calculées à l'étape 1 ci-dessus (qui elles-mêmes dépendent des dates de fin d'exécution possibles de toutes les tâches précédentes) et du nombre de ses durées d'exécution possibles. Nous calculons l'ensemble des dates de fin possibles de l'ensemble des tâches prédécesseurs de t,

$s^1_{t_{10}}$=$max(I^-_{t_{10}}, max(e^1_{t_6}, e^1_{t_7}))$=$max(20, max(14, 20)) = 20$	$e^{11}_{t_{10}} = s^1_{t_{10}} + d^1_{t_{10}} = 20 + 4 = 24$
$s^2_{t_{10}}$=$max(I^-_{t_{10}}, max(e^1_{t_6}, e^2_{t_7}))$=$max(20, max(14, 22)) = 22$	$e^{12}_{t_{10}} = s^1_{t_{10}} + d^2_{t_{10}} = 20 + 6 = 26$
$s^3_{t_{10}}$=$max(I^-_{t_{10}}, max(e^2_{t_6}, e^1_{t_7}))$=$max(20, max(15, 20)) = 20$	$e^{13}_{t_{10}} = s^1_{t_{10}} + d^3_{t_{10}} = 20 + 10 = 30$
$s^4_{t_{10}}$=$max(I^-_{t_{10}}, max(e^2_{t_6}, e^2_{t_7}))$=$max(20, max(15, 22)) = 22$	$e^{21}_{t_{10}} = s^2_{t_{10}} + d^1_{t_{10}} = 22 + 4 = 26$
$s^5_{t_{10}}$=$max(I^-_{t_{10}}, max(e^3_{t_6}, e^1_{t_7}))$=$max(20, max(16, 20)) = 20$	$e^{22}_{t_{10}} = s^2_{t_{10}} + d^2_{t_{10}} = 22 + 6 = 28$
$s^6_{t_{10}}$=$max(I^-_{t_{10}}, max(e^3_{t_6}, e^2_{t_7}))$=$max(20, max(16, 22)) = 22$	$e^{23}_{t_{10}} = s^2_{t_{10}} + d^3_{t_{10}} = 22 + 10 = 32$
$S_{t_{10}} = \{20, 22\}$	$E_{t_{10}} = \{24, 26, 28, 30, 32\}$
$\mathcal{I}_{t_{10}} = \{[20, 24], [20, 26], [20, 30], [22, 26], [22, 28], [22, 32]\}$	

TAB. 5.1 – Calcul des intervalles d'exécution possibles de la tâche t_{10} de l'exemple de la section 5.1.5

noté $E(t_1, \ldots, t_n)$ comme suit :

$$E(t_1, \ldots, t_n) = \{max(e^{k_{t_1}}_{t_1}, e^{k_{t_2}}_{t_2}, \ldots, e^{k_{t_n}}_{t_n}) / 1 \leq k_{t_1} \leq p_{t_1} \ldots 1 \leq k_{t_n} \leq p_{t_n}\}$$

où $i = 1..n$ et p_{t_i} représente la cardinalité de l'ensemble des dates de fin d'exécution possibles de t_i. Par exemple si une tâche t a trois prédécesseurs directs t_1, t_2 et t_3 qui terminent leurs exécutions à $E_{t_1} = \{e^1_{t_1}, e^2_{t_1}, e^3_{t_1}\}$, $E_{t_2} = \{e^1_{t_2}, e^2_{t_2}, e^3_{t_2}\}$ et $E_{t_3} = \{e^1_{t_3}, e^2_{t_3}\}$ respectivement, l'ensemble $E(t_1, t_2, t_3)$ est composé de l'ensemble à 18 éléments au plus des dates de fin d'exécution possibles suivant :

$$\{max(e^1_{t_1}, e^1_{t_2}, e^1_{t_3}), max(e^1_{t_1}, e^1_{t_2}, e^2_{t_3}), \ldots\}$$

Au pire des cas, c.-à-d. si tous les maximums de toutes ces combinaisons sont différents, nous avons :

$$|E(t_1, \ldots, t_n)| = |E_{t_1}| \cdot |E_{t_2}| \cdot \ldots \cdot |E_{t_n}|$$

Une fois déterminé l'ensemble des dates de fin possibles ($E(t_1, \ldots, t_n)$) pour tous les prédécesseurs de la tâche t, nous calculons son nombre d'intervalles d'exécution possibles. Au pire des cas, ce nombre est égal à $|E(t_1, \ldots, t_n)| \cdot |\Delta_t|$.

5.2.3.3 Admissibilité des plans

Un plan *faisable* $\mathcal{P}_f \in \mathcal{P}_F$ est *admissible* s'il vérifie toutes les contraintes temporelles. Plus formellement : Pour toute tâche $t \in T(\mathcal{P}_f)$, il existe $e^i_t \in E_t$ telle que $e^i_t \leq I^+_t$. Si une date de fin de l'ensemble E_t dépasse I^+_t, on considère que l'intervalle d'exécution $I^i_t = [s^i_t, e^i_t]$ n'est pas valide. Par contre si toutes les dates de fin e^i_t de t dépassent I^+_t, on considère que la tâche t ne peut pas être exécutée et par suite le plan n'est pas admissible. Cette étape nous permet d'éliminer les plans qui contiennent une ou plusieurs tâches dont l'exécution est impossible à cause de la non-validité des intervalles d'exécution possibles.

Exemple 5 *Imaginons que le calcul des intervalles d'exécution possibles de la tâche* t_{16} : $\langle[42,55],\{10,11,12,13\},\{0.5,0.3,0.2,0.1\},\{40,45,45,55\}\rangle$ *du plan faisable* $\mathcal{P}_f = [[t_2 \rightarrow t_7],[t_1 \rightarrow t_6]] \rightarrow t_{10} \rightarrow t_{14} \rightarrow t_{16} \rightarrow t_{18}$ *donne le résultat suivant :*

$$\mathcal{I}_{t_{16}} = \{[46,56],[46,57],[46,58],[56,59],[50,60],[50,61],[50,62],[50,63]\}$$

Puisque toutes les dates de fin des ces intervalles d'exécution dépassent la date de fin au plus tard de t_{16} *initialisée à* 55*, nous considérons que tous ces intervalles ne sont pas valides et que la tâche* t_{16} *ne peut pas s'exécuter dans ce plan faisable. Par conséquent ce plan n'est pas admissible et nous ne l'analysons plus dans les étapes suivantes.*
Notons que la même tâche peut être valide dans un autre plan faisable, $\mathcal{P}_f = t_2 \rightarrow t_7 \rightarrow t_{12} \rightarrow t_{16} \rightarrow t_{18}$ *par exemple, s'il existe* $e^i_{t_{16}} \in E_{t_{16}}$ *telle que* $e^i_{t_{16}} \leq I^+_{t_{16}}$.

Nous désignons par $\mathcal{P}_A$ l'ensemble de tous les plans admissibles. Nous avons $\mathcal{P}_A = \bigcup \mathcal{P}_a$, tel que $\mathcal{P}_a$ est un plan admissible.

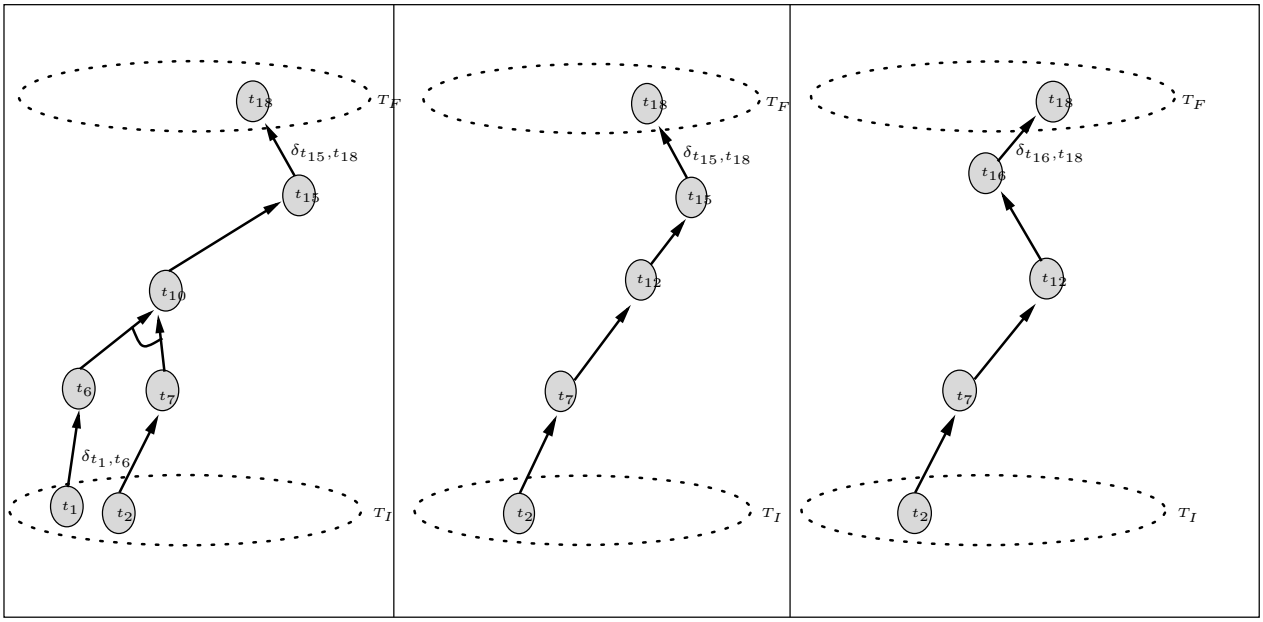

FIG. 5.11 – Les plans admissibles sélectionnés de l'exemple 1

Exemple 6 *Dans notre exemple,* $\mathcal{P}_A = \{\mathcal{P}_{a_1}, \mathcal{P}_{a_2}, \mathcal{P}_{a_3}\}$ *tels que :*

- $\mathcal{P}_{a_1} = [[t_2 \rightarrow t_7],[t_1 \rightarrow t_6]] \rightarrow t_{10} \rightarrow t_{15} \rightarrow t_{18}$
- $\mathcal{P}_{a_2} = t_2 \rightarrow t_7 \rightarrow t_{12} \rightarrow t_{16} \rightarrow t_{18}$
- $\mathcal{P}_{a_3} = t_2 \rightarrow t_7 \rightarrow t_{12} \rightarrow t_{15} \rightarrow t_{18}$

Ces plans sont représentés dans la figure 5.11.

5.3 Conclusion

Dans ce chapitre, nous avons expliqué la méthode utilisée pour extraire du graphe ET/OU les plans temporellement admissibles. Tout d'abord, nous recherchons les plans faisables, c'est-à-dire les plans qui respectent les contraintes temporelles qualitatives (symboliques). Pour ce faire, nous recherchons en chaînage arrière dans le graphe ET/OU tous les chemins amenant aux buts. Ensuite, nous propageons les plans faisables obtenus pour en extraire les plans admissibles c'est-à-dire les plans qui respectent les contraintes temporelles quantitatives (numériques). Nous calculons pour chaque tâche d'un plan faisable, les intervalles d'exécution possibles et nous vérifions leur validité par rapport à la fenêtre temporelle de la tâche. Nous avons aussi étudié le nombre possible d'intervalles d'exécution. Ce nombre dépend, en général, de la date de début au plus tôt et de la date de fin au plus tard de la tâche, de son nombre de durées d'exécution possibles et des dates de fin d'exécution possibles de ses tâches précédentes.

Ayant généré les plans temporellement admissibles du graphe ET/OU, l'étape suivante est de sélectionner parmi eux un seul plan-solution. Cette sélection dépend des contraintes à savoir le coût et la probabilité. Cela nous conduit à un problème de planification multi-critères. Nous choisissons, en fonction de préférences de l'utilisateur, le meilleur plan (plan-solution) pouvant être exécuté. Le choix d'un tel plan est, en général, un compromis entre les différents critères entrant en jeu (temps, coût et probabilité).

Dans le chapitre suivant, nous faisons la propagation de probabilités sur les intervalles d'exécution possibles et expliquons la méthode de calcul des valeurs espérées de coût et de temps des tâches des plans admissibles. Dans le chapitre 7, nous sélectionnons le meilleur plan-solution selon les critères de l'utilisateur. Le plan-solution à proposer pour atteindre le but respecte toutes les contraintes et répond au mieux aux attentes de l'utilisateur.

Chapitre 6

Calcul d'une fonction objective

La recherche des plans admissibles peut en fournir plusieurs. Pour en choisir un seul pour être exécuté, plusieurs critères de préférences (probabilité, temps et coût) sont utilisés pour analyser et comparer les plans.

Nous avons étudié plusieurs méthodes de sélection du meilleur plan-solution. Dans cet objectif, le planificateur calcule tout d'abord les valeurs espérées de coût et de temps des tâches. Ensuite, il choisit selon les préférences de l'utilisateur le plan qui répond le plus à ses attentes. Les différentes méthodes de sélection sont l'objectif du chapitre 7. Cependant, ce chapitre est consacré à la propagation de probabilités sur les intervalles d'exécution et le calcul des valeurs espérées de coût et de temps des tâches.

Rappelons qu'à chaque tâche t sont associés un ensemble de durées $\Delta_t = \{d_t^1, d_t^2, ..., d_t^m\}$, un ensemble de probabilités $Pr_t = \{pr_t^1, pr_t^2, ..., pr_t^m\}$, où pr_t^i est la probabilité que la tâche t s'exécute pendant la durée d_t^i et un ensemble de coûts $C_t = \{c_t^1, c_t^2, ..., c_t^m\}$, où c_t^i est le coût de l'exécution de la tâche t pendant la durée d_t^i. Rappelons aussi que dans chaque plan admissible, les tâches sont reliées par des relations de précédence indiquant un ordre d'exécution entre elles.

Nous présentons dans la section suivante l'algorithme de propagation de probabilités pour calculer les probabilités sur les intervalles d'exécution possibles calculés dans la section 5.2.3 du chapitre 5. Ensuite, dans la section 6.2, nous montrons comment calculer les valeurs espérées du coût et du temps des tâches des plans admissibles.

6.1 Propagation de probabilités

Nous décrivons dans cette section comment attribuer une probabilité à chacun des intervalles d'exécution possibles calculés précédemment. La probabilité d'un intervalle d'exécution I_t^i d'une tâche t dépend de la date à laquelle la tâche commence (date de fin de la tâche précédente) et de la probabilité de la durée d'exécution pr_t^i. Pour simplifier, nous supposons que les dates de fin utilisées dans les formules sont les dates des fins possibles avec les délais entre les tâches

(cette hypothèse a pour but d'éviter la répétition du terme "délai" dans les formules). Soient I_t^i le $i^{\text{ème}}$ intervalle de la tâche t, s_t^i sa $i^{\text{ème}}$ date de début et e_t^i sa $i^{\text{ème}}$ date de fin ($I_t^i = [s_t^i, e_t^i]$). La probabilité qu'une tâche t s'exécute dans l'intervalle I_t^i est égale à la probabilité que son exécution commence à s_t^i et se termine à e_t^i. Autrement dit, l'exécution de t commence à s_t^i et dure $e_t^i - s_t^i = d_t^i$ unités de temps. La probabilité que t commence son exécution à la date s_t^i dépend des dates de fin possible de la (ou des) tâche(s) qui la précède (précèdent).

En fait, nous calculons tout d'abord les probabilités d'exécution sur les dates de début des tâches dans les plans admissibles ensuite nous calculons les probabilités sur les intervalles d'exécution en utilisant les résultats de la première étape.

Dans la section suivante, nous présentons l'algorithme permettant de calculer les probabilités sur les dates de début d'exécution de chaque tâche.

6.1.1 Calcul des probabilités sur les dates de début

Soient une tâche t appartenant à l'ensemble des tâches d'un plan admissible $\mathcal{P}_a$ ($t \in T(\mathcal{P}_a)$), $Pr_t = \{pr_t^1, pr_t^2, ..., pr_t^m\}$ l'ensemble de probabilités où pr_t^i est la probabilité que t s'exécute pendant la durée d_t^i et $\mathcal{I}_t = \{I_t^i = [s_t^i, e_t^i],\ i = 1..m_t$ (m_t est le nombre d'intervalles d'exécution possibles de t)$\}$ l'ensemble des intervalles d'exécution possibles de t calculés dans la section 5.2.3 du chapitre 5.

6.1.1.1 Cas d'une tâche initiale

Les tâches initiales d'un plan admissible n'ont pas de prédécesseurs, par conséquent leurs probabilités d'exécution ne dépendent que de leurs durées d'exécution.

La probabilité que l'exécution d'une tâche initiale t commence à s_t, notée $pr_{debut}(s_t)$, est égale à 1. En effet, tous les intervalles d'exécution valides de t ont une date de début s_t supérieure ou égale à la date de début ($start\_time$) proposée par l'agent. C'est-à-dire que tous les intervalles d'exécution peuvent être exécutés pendant leurs durées possibles d'exécution et avec les probabilités d'exécution respectives.

6.1.1.2 Cas d'une tâche intermédiaire ou finale

Afin de commencer l'exécution d'une tâche intermédiaire ou finale t, tous les prédécesseurs de cette dernière doivent avoir été exécutés. L'exécution de t durant un intervalle d'exécution $I_t^i = [s_t^i, e_t^i]$ dépend de sa date de début s_t^i et des dates de fin de ses prédécesseurs. La probabilité que l'exécution de t commence à s_t^i est définie de la façon suivante :

Cas d'une tâche *OU*

Rappelons qu'une tâche *OU* dans un plan admissible est une tâche qui n'a qu'un seul prédécesseur. Si t a un seul prédécesseur direct t' ($t' \rightarrow t$) dans le plan admissible $\mathcal{P}_a$, si $E_{t'} = \{e^1_{t'}, ..., e^p_{t'}\}$ et $S_t = \{s^1_t, s^2_t, ..., s^{m_t}_t\}$ est l'ensemble des dates de début de la tâche t, alors la probabilité que l'exécution de t commence à s^r_t sachant que son prédécesseur t' a terminé son exécution à $e^j_{t'}$, notée $pr_{debut}(s^r_t|e^j_{t'})$, est égale à zéro si s^r_t est inférieure à $e^j_{t'}$ et elle est égale à un sinon.

Cas d'une tâche *ET*

Soit t a un ensemble de prédécesseurs directs $t_1, t_2, \ldots, t_n$ dans le plan admissible $\mathcal{P}_a$, c.à.d. si $[t_1, t_2, \ldots, t_n] \rightarrow t$ alors soit $t_i \in \{t_1, t_2, \ldots, t_n\}$ telle que $E_{t_i} = \{e^1_{t_i}, e^2_{t_i}, \ldots, e^{j_i}_{t_i}\}$ où j_i représente le nombre de dates de fin de la tâche t_i. Soit $S_t = \{s^1_t, s^2_t, ..., s^{m_t}_t\}$ l'ensemble des dates de début possibles de la tâche t. La probabilité que l'exécution de t commence à s^r_t ($r = 1...m_t$) sachant que les prédécesseurs $t_i \in \{t_1, t_2, \ldots, t_n\}$ de t ont terminé leur exécution respectivement à $e^{k_{t_i}}_{t_i}$ ($k_{t_i} = 1..j_i$) est égale à zéro si s^r_t est inférieure à $max\{e^{k_{t_1}}_{t_1}, e^{k_{t_2}}_{t_2}, \ldots, e^{k_{t_n}}_{t_n}\}$ et elle est égale à un sinon.

6.1.2 Calcul des probabilités sur les intervalles d'exécution

Les probabilités sur les intervalles d'exécution dépendent des probabilités sur les durées d'exécution et celles sur les dates de début d'exécution.

Soient une tâche t appartenant à l'ensemble des tâches d'un plan admissible $\mathcal{P}_a$ ($t \in T(\mathcal{P}_a)$), $Pr_t = \{pr^1_t, pr^2_t, ..., pr^m_t\}$ l'ensemble de probabilités où pr^r_t est la probabilité que t s'exécute pendant la durée d^r_t. Soit $\mathcal{I}_t = \{I^i_t = [s^i_t, e^i_t]$ où $i = 1..m_t$ (m_t est le nombre d'intervalles d'exécution possibles)$\}$ l'ensemble des intervalles d'exécution possible de t. On note par :

- $pr_t(d^i_t|s^i_t)$ la probabilité que l'exécution de t dure d^i_t unités de temps si elle a commencé à s^i_t,
- $pr_{debut}(s_t)$ la probabilité que t commence son exécution à la date de début s_t dans le cas où elle est une tâche initiale,
- $pr_{debut}(s^r_t|e^j_{t'})$ la probabilité que t commence son exécution à la date de début s^r_t dans le cas où elle est une tâche *OU*,
- $pr_{debut}(s^r_t|e^{k_{t_i}}_{t_i})$ la probabilité que t commence son exécution à la date de début s^r_t dans le cas où elle est une tâche *ET*.

Dans la suite, nous calculons les probabilités sur les intervalles d'exécution dans les cas de tâches initiales, intermédiaires ou finales et aussi selon les deux types de tâches : *OU* et *ET*.

6.1.2.1 Cas d'une tâche initiale

Les tâches initiales ne dépendent pas des prédécesseurs et par suite la probabilité qu'une tâche initiale t s'exécute dans un intervalle I_t^r est égale à la probabilité que cette tâche commence son exécution à s_t et dure d_t^r unités de temps. Plus formellement :

$$Pr(I_t^r) = pr_{debut}(s_t) \cdot pr_t(d_t^r|s_t) \tag{6.1}$$

6.1.2.2 Cas d'une tâche intermédiaire ou finale

L'exécution d'une tâche intermédiaire ou finale t dépend des dates de fin de tous ses prédécesseurs. Les probabilités sur les intervalles d'exécution de t sont calculées de la façon suivante :

Cas d'une tâche *OU*

La probabilité qu'une tâche t ayant un seul prédécesseur t' s'exécute durant un intervalle I_t^r sachant que son prédécesseur a terminé son exécution à $e_{t'}^j$ est notée $Pr(I_t^r|e_{t'}^j)$. Elle est définie de la manière suivante :

$$Pr(I_t^r|e_{t'}^j) = pr_{debut}(s_t^r|e_{t'}^j) \cdot pr_t(d_t^r|s_t^r) \tag{6.2}$$

où $e_{t'}^j$ est la date de fin de la dernière tâche exécutée (prédécesseur direct). En effet, une tâche t ne peut commencer à être exécutée que lorsque son prédécesseur t' est exécuté. La date de début de t dépend donc de la date de fin de t'.

Exemple 7 *Dans notre exemple de la section 5.1.5 du chapitre 5, la probabilité que l'exécution de la tâche t_7, ayant un seul prédécesseur t_2 dans le plan admissible $\mathcal{P}_a = [[t_2 \rightarrow t_7], [t_1 \rightarrow t_6]] \rightarrow t_{10} \rightarrow t_{15} \rightarrow t_{18}$, dure 15 unités de temps lorsqu'elle commence à 5, est égale à 0.5. La probabilité que t_7 s'exécute durant l'intervalle $[5, 20]$ est alors 0.5. En effet, $Pr_{t_7}(d_{t_7}^1 = 15|s_{t_7}^1 = 5) = 0.5$ et $pr_{debut}(s_{t_7}^1 = 5|e_{t_2}^1 = 4) = 1$ et par conséquent $Pr(I_{t_7}^1|e_{t_2}^1) = 1 \cdot 0.5 = 0.5$.*
Si par contre, la tâche t_2 se termine à 6, on aura : $pr_{debut}(s_{t_7}^1 = 5|e_{t_2}^1 = 6) = 0$ et par suite $Pr([5, 20]|6) = 0 \cdot 0.5 = 0$.

Cas d'une tâche *ET*

Dans le cas où t a $\Delta_t = \{d_t^1, d_t^2, ..., d_t^m\}$ et un ensemble de prédécesseurs directs $t_1, t_2, \ldots, t_n$, la probabilité que t s'exécute durant un intervalle I_t^r sachant que chaque prédécesseur $t_i \in \{t_1, t_2, \ldots, t_n\}$ termine son exécution à $e_{t_i}^{k_{t_i}}$ ($k_{t_i} = 1..j_i$ où j_i représente le nombre de dates de fin de la tâche t_i), est définie comme suit :

$$Pr(I_t^r|e_{t_i}^{k_{t_i}}) = pr_{debut}(s_t^r|e_{t_i}^{k_{t_i}}) \cdot pr_t(d_t^r|s_t^r) \tag{6.3}$$

où $r = 1..m$.

Exemple 8 *La probabilité que l'exécution de la tâche t_{10}, ayant deux prédécesseurs t_6 et t_7 dans le plan admissible $\mathcal{P}_a = [[t_2 \rightarrow t_7], [t_1 \rightarrow t_6]] \rightarrow t_{10} \rightarrow t_{15} \rightarrow t_{18}$, dure 4 unités de temps lorsqu'elle commence à 20, est égale à 0.1. La probabilité que t_{10} s'exécute durant l'intervalle $[20, 24]$ sachant que t_6 se termine à 14 et t_7 à 20 est égale à 0.1. En effet, $Pr_{t_{10}}(d^1_{t_{10}} = 4|s^1_{t_{10}} = 20) = 0.1$ et $pr_{debut}(s^1_{t_{10}} = 20|(e^1_{t_6} = 14, e^1_{t_7} = 20)) = 1$ et par suite $Pr([20, 24]|(14, 20)) = 1 \cdot 0.1 = 0.1$.*
Par contre, la probabilité que t_{10} s'exécute durant l'intervalle $[20, 24]$ sachant que t_6 se termine à 14 et t_7 à 22 est égale à 0. En effet, $Pr_{t_{10}}(d^1_{t_{10}} = 4|s^1_{t_{10}} = 20) = 0.1$ et $pr_{debut}(s^1_{t_{10}} = 20|(e^1_{t_6} = 14, e^2_{t_7} = 22)) = 0$ et par suite $Pr([20, 24]|(14, 22)) = 0 \cdot 0.1 = 0$. Cela signifie que cet intervalle n'est pas exécutable si t_6 termine son exécution à la date 14 et t_7 à la date 22.

Pour simplifier, nous notons, que ce soit le type de la tâche t, la probabilité d'exécution de t durant un intervalle d'exécution possible I^r_t par $Pr(I^r_t)$.

6.2 Valeurs espérées de coût et de temps

La valeur espérée de coût représente une moyenne (espérance) entre le coût et la probabilité. La valeur espérée de temps représente une moyenne (espérance) entre la durée d'exécution et la probabilité. Pour préférer un plan aux autres, il faut qu'il ait une valeur espérée minimale pour pouvoir réduire le coût et le temps tout en ayant une grande probabilité d'exécution. Pour en choisir un seul, nous calculons la valeur espérée de coût et de temps de toutes les tâches des plans admissibles. Ensuite, nous calculons les valeurs espérées totales de coût et de temps de chaque plan admissible en fonction des valeurs espérées de ses tâches.

6.2.1 Calcul des valeurs espérées de coût et de temps d'une tâche

Pour chaque tâche dans un plan admissible $\mathcal{P}_a$, nous calculons les valeurs espérées de coût et de temps. La valeur espérée de coût est calculée en fonction des coûts associés aux durées d'exécution de la tâche et des probabilités associées à ses intervalles d'exécution (calculées dans la section 6.1.2). La valeur espérée de temps est quant à elle calculée en fonction des durées possibles d'exécution de la tâche et des probabilités associées à ses intervalles d'exécution.

Soit une tâche t appartenant à l'ensemble des tâches d'un plan admissible $\mathcal{P}_a$ ($t \in T(\mathcal{P}_a)$). Soit $\Delta_t = \{d^1_t, d^2_t, ..., d^m_t\}$ l'ensemble des durées d'exécution possibles de t. Soit $C_t = \{c^1_t, c^2_t, ..., c^m_t\}$ l'ensemble des coûts d'exécution de t, où c^r_t est le coût de l'exécution de la tâche t pendant la durée d^r_t ($r = 1..m$).

6.2.1.1 Cas d'une tâche initiale

La valeur espérée de coût d'une tâche initiale t est égale à la somme des produits des probabilités de chaque intervalle d'exécution possible de t par le coût associé à la durée de l'intervalle d'exécution. Plus formellement, la valeur espérée de coût de t est calculée de la façon suivante :

$$\vartheta(cout(t)) = \sum_{r=1}^{m} Pr(I_t^r) \cdot c_t^r \tag{6.4}$$

La valeur espérée de temps d'une tâche initiale t est égale à la somme des produits des probabilités de chaque intervalle d'exécution possible de t par la durée de l'intervalle d'exécution. Plus formellement, la valeur espérée de temps de t est calculée de la façon suivante :

$$\vartheta(temps(t)) = \sum_{r=1}^{m} Pr(I_t^r) \cdot d_t^r \tag{6.5}$$

Exemple 9 *La valeur espérée de coût de la tâche t_1 du plan admissible $\mathcal{P}_a = [[t_2 \rightarrow t_7], [t_1 \rightarrow t_6]] \rightarrow t_{10} \rightarrow t_{15} \rightarrow t_{18}$ de notre exemple, est égale à $0.2 \cdot 12 + 0.8 \cdot 20 = 18.4$. Sa valeur espérée de temps est égale à $0.2 \cdot 5 + 0.8 \cdot 6 = 5.8$.*

6.2.1.2 Cas d'une tâche intermédiaire ou finale

Les valeurs espérées de coût et de temps d'une tâche intermédiaire ou finale dépendent des probabilités des intervalles d'exécution possibles de la tâche et soit des coûts associés à ses durées d'exécution soit des durées d'exécution possibles de la tâche.

Cas d'une tâche *OU*

Si t a un seul prédécesseur t' dont l'ensemble des dates de fin possibles est $E_{t'} = \{e_{t'}^1, e_{t'}^2, \ldots, e_{t'}^p\}$, alors soit $Pr(I_t^r | e_{t'}^j)$ (section 6.1.2) la probabilité que t s'exécute pendant une durée d_t^r sachant que son prédécesseur t' termine son exécution à $e_{t'}^j$. Nous calculons les valeurs espérées de coût et de temps comme suit :

$$\vartheta(cout(t|t')) = \sum_{j=1}^{p} \sum_{r=1}^{m} Pr(I_t^r | e_{t'}^j) \cdot c_t^r \tag{6.6}$$

$$\vartheta(temps(t|t')) = \sum_{j=1}^{p} \sum_{r=1}^{m} Pr(I_t^r | e_{t'}^j) \cdot d_t^r \tag{6.7}$$

Exemple 10 *La valeur espérée de coût de la tâche t_6 ayant un seul prédécesseur t_1 dans le plan admissible $\mathcal{P}_a = [[t_2 \rightarrow t_7], [t_1 \rightarrow t_6]] \rightarrow t_{10} \rightarrow t_{15} \rightarrow t_{18}$ de notre exemple, est égale à $0.3 \cdot 8 + 0.3 \cdot 10 + 0.4 \cdot 20 = 13.4$. Sa valeur espérée de temps est égale à $0.3 \cdot 8 + 0.3 \cdot 9 + 0.4 \cdot 10 = 9.1$. Les valeurs espérées de coût et de temps de la tâche t_7 ayant un seul prédécesseur t_2 dans le même plan admissible sont égales à 85 et 32 respectivement.*

Cas d'une tâche *ET*

Si t a un ensemble de prédécesseurs directs $t_1, t_2, \ldots, t_n$, alors soit $E_{t_i} = \{e^1_{t_i}, e^2_{t_i}, \ldots, e^{j_i}_{t_i}\}$ l'ensemble des dates de fin possibles de $t_i \in \{t_1, t_2, \ldots, t_n\}$ alors :

la valeur espérée de coût de t est calculée de la façon suivante :

$$\vartheta(cout(t|t_1, \ldots, t_n)) = \sum_{i=1}^{n} \vartheta(cout(t|t_i)) \tag{6.8}$$

la valeur espérée de temps de t est calculée de la façon suivante :

$$\vartheta(temps(t|t_1, \ldots, t_n)) = max_{i=1}^{n} \vartheta(temps(t|t_i)) \tag{6.9}$$

Exemple 11 *La valeur espérée de coût de la tâche t_{10} ayant deux prédécesseurs t_6 et t_7 dans le plan admissible $\mathcal{P}_a = [[t_2 \rightarrow t_7], [t_1 \rightarrow t_6]] \rightarrow t_{10} \rightarrow t_{15} \rightarrow t_{18}$ de notre exemple, est égale à $0.1 \cdot 2 + 0.3 \cdot 8 + 0.6 \cdot 20 + 0.1 \cdot 2 + 0.1 \cdot 2 + 0.3 \cdot 8 + 0.3 \cdot 8 = 19.8$.*

Pour simplifier, nous notons, que ce soit le type de la tâche t, la valeur espérée de coût de t par $\vartheta(cout(t))$ et la valeur espérée de temps de t par $\vartheta(temps(t))$.

6.3 Conclusion

La propagation de probabilités dans les plans admissibles du graphe ET/OU permet de calculer les probabilités sur les intervalles d'exécution des tâches. Ces probabilités dépendent des probabilités des durées d'exécution et des probabilités que les tâches prédécesseurs aient terminé leur exécution. Les probabilités sur les intervalles d'exécution permettent de calculer les valeurs espérées de coût et de temps des tâches, pour ensuite les utiliser afin de sélectionner un seul plan-solution. Le calcul de ces valeurs varie selon que la tâche est une tâche initiale, intermédiaire ou finale et aussi selon les deux types de tâches : tâches *OU* et tâches *ET*. La dernière étape de sélection du plan-solution est expliquée dans le chapitre suivant.

Chapitre 7

Sélection multi-critères

Choisir le meilleur plan parmi tous ceux qui sont temporellement admissibles est une étape très importante. C'est ici que les préférences de l'utilisateur entrent en jeu. Comme nous travaillons dans un environnement incertain, il est difficile de trouver le meilleur plan au sens vrai du mot. Il s'agit plutôt d'un plan *préféré* selon un ou plusieurs critères. C'est pourquoi nous offrons à l'utilisateur la possibilité d'indiquer ses critères de choix du meilleur plan selon ses attentes. Le planificateur en tient compte et lui propose un plan *conseillé* pour l'exécution afin de réaliser l'objectif initial tout en respectant toutes les contraintes et répondant à ses attentes.

Dans ce chapitre, nous exposons les méthodes utilisées par notre planificateur afin de sélectionner le meilleur plan parmi tous ceux qui sont admissibles. La sélection du meilleur plan se fait en fonction des probabilités, valeurs espérées, coûts et durées. Une fois le meilleur plan est sélectionné, le planificateur propose à l'utilisateur le meilleur ordonnancement possible pour les tâches du plan en question.

Dans ce qui suit, nous présentons tout d'abord les méthodes de sélection d'un plan-solution. Ensuite nous donnons la définition d'un ordonnancement et détaillons les méthodes de sélection du meilleur ordonnancement.

7.1 Sélection du meilleur plan

Soit $\mathcal{P}_A$ l'ensemble des plans admissibles sélectionnés dans la section 5.2.2 du chapitre 5. Rappelons qu'un plan admissible est un chemin dans le graphe ET/OU qui respecte les contraintes temporelles qualitatives et quantitatives. Chaque plan admissible $\mathcal{P}_a \in \mathcal{P}_A$ est composé d'un ensemble de tâches $t_1, t_2, \ldots, t_n$ reliées par des relations de précédence indiquant un ordre d'exécution entre elles et dont leurs réalisations entraînent la réalisation du but initial. Chaque tâche $t_i \in T(\mathcal{P}_a)$ $(i = 1..n)$ possède un ensemble d'intervalles d'exécution possibles $\mathcal{I}_{t_i} = \{I^1_{t_i}, I^2_{t_i}, \ldots, I^m_{t_i}\}$ tel que $I^j_{t_i} = [s^j_{t_i}, e^j_{t_i}]$ où $s^j_{t_i}$ et $e^j_{t_i}$ sont respectivement une date de début et une date de fin d'exécution possibles de t_i avec une probabilité d'exécution $Pr(I^j_{t_i})$ où $j = 1..m$. Rappelons aussi

que la valeur espérée de coût de t_i est notée $\vartheta(cout(t_i))$ et la valeur espérée de temps de t_i est notée $\vartheta(temps(t_i))$.

7.1.1 Sélection selon la probabilité

Le plan-solution le plus probable [Baki *et al.* 04, Baki & Beynier 04], est celui ayant la plus grande probabilité. Afin de choisir le plan le plus probable, le planificateur calcule les probabilités de tous les plans admissibles et sélectionne celui ayant la probabilité la plus élevée.

La probabilité d'exécution d'une tâche est égale au produit de toutes les probabilités de ses intervalles d'exécution possibles. La probabilité d'exécution d'un plan admissible est égale à la somme de toutes les probabilités d'exécution de ses tâches. Ainsi, la probabilité d'exécution d'un plan est égale au produit cartésien de tous les intervalles d'exécution de toutes ses tâches. Plus formellement :

$$Pr(\mathcal{P}_a) = \sum_{i=1}^{n} \Pi_{j=1}^{m} Pr(I_{t_i}^j) \tag{7.1}$$

Nous notons par $\mathcal{P}^s$ le plan-solution choisi pour être exécuté. Si l'utilisateur souhaite avoir le plan le plus probable en respectant toutes les contraintes de l'environnement, il choisit celui dont la probabilité est la plus élevée :

$$\begin{aligned} \mathcal{P}^s \quad = \quad & argmax \quad Pr(\mathcal{P}_a) \\ & \mathcal{P}_a \in \mathcal{P}_A \end{aligned} \tag{7.2}$$

7.1.2 Sélection à base d'une seule fonction de valeur espérée

Si l'utilisateur souhaite avoir un compromis entre une grande probabilité d'exécution et un coût et temps réduits [Baki & Bouzid 05b, Baki 05], le planificateur tient compte dans la sélection du plan-solution des valeurs espérées de coût et de temps des tâches calculées dans la section 6.2 du chapitre 6. Nous présentons dans ce qui suit la méthode de calcul des valeurs espérées de coût et de temps des plans admissibles. Ensuite, nous sélectionnons le meilleur plan-solution en utilisant une fonction de valeur espérée valorisant l'importance qu'attache l'utilisateur au temps et au coût.

La valeur espérée totale de coût d'un plan admissible $\mathcal{P}_a$, notée $\vartheta(cout(\mathcal{P}_a))$, est calculée en additionnant toutes les valeurs espérées de coût (section 6.2.1 du chapitre 6) de toutes les tâches de ce plan. Plus formellement :

$$\vartheta(cout(\mathcal{P}_a)) = \sum_{i=1}^{n} \vartheta(cout(t_i)) \tag{7.3}$$

tel que n est le nombre des tâches dans le plan $\mathcal{P}_a$.

La valeur espérée totale de temps d'un plan admissible $\mathcal{P}_a$ (initialisée à $max(\vartheta(temps(t)))$ où $t \in T_I$) est calculée selon le type de la tâche courante comme suit :

– si t_i est une tâche OU et a un seul prédécesseur t_j ($t_j \rightarrow t_i$) avec une valeur espérée $\vartheta(temps(t_j))$, alors on ajoute $\vartheta(temps(t_j))$ à la valeur espérée totale du plan. Plus formellement :

$$\vartheta(temps(\mathcal{P}_a)) = \vartheta(temps(\mathcal{P}_a)) + \vartheta(temps(t_j)) \tag{7.4}$$

– si t_i est une tâche ET et a un ensemble de prédécesseurs $t_1, t_2, \ldots, t_r$ ($[t_1, t_2, \ldots, t_r] \rightarrow t_i$) alors, on ajoute le maximum de la valeur espérée des tâches $t_1, t_2, \ldots, t_r$ à la valeur espérée totale du plan. Plus formellement :

$$\vartheta(temps(\mathcal{P}_a)) = \vartheta(temps(\mathcal{P}_a)) + max_{k=1}^{r}\vartheta(temps(t_k)) \tag{7.5}$$

Pour obtenir la valeur espérée totale du plan, nous additionnons ses valeurs espérées de temps et de coût. Nous pondérons cette somme par des coefficients pour permettre d'ajuster l'importance relative aux différentes valeurs espérées selon les préférences de l'utilisateur. Nous traduisons ces préférences en une fonction $\vartheta(\mathcal{P}_a)$: ϑ où ϑ est la fonction valeur espérée ayant comme coefficients α pour le coût et β pour le temps. Ces coefficients expriment l'importance qu'accorde l'utilisateur au coût ou au temps. L'équation obtenue représente un problème d'analyse multi-critères. Plus formellement, cette valeur espérée est exprimée comme suit :

$$\vartheta(\mathcal{P}_a) = \alpha \cdot \vartheta(cout(\mathcal{P}_a)) + \beta \cdot \vartheta(temps(\mathcal{P}_a)) \tag{7.6}$$

tels que $\alpha + \beta = 1$, $0 \leq \alpha \leq 1$, $0 \leq \beta \leq 1$ et $\mathcal{P}_a \in \mathcal{P}_A$. Dans le cas où l'utilisateur n'a pas de préférence entre le coût et le temps, les coefficients α et β ont la même valeur (0.5). Dans le cas où il préfère un critère par rapport à l'autre, le coefficient associé à celui-ci est plus élevé que l'autre (en gardant toujours $\alpha + \beta = 1$).

Parmi tous les plans admissibles, nous choisissons pour l'exécution le plan qui a la valeur espérée minimale :

$$\begin{aligned} \mathcal{P}^s \;=\; & \underset{\mathcal{P}_a \in \mathcal{P}_A}{argmin} \quad \vartheta(\mathcal{P}_a) \end{aligned} \tag{7.7}$$

7.1.3 Sélection à base d'un ordre lexicographique

Lorsque plusieurs plans répondent aux critères de préférence de l'utilisateur, il donne une priorité à un seul critère par rapport aux autres. Dans ce qui suit, nous présentons certaines méthodes de choix d'un seul plan-solution selon un ordre lexicographique exprimé sous la forme des priorités données aux probabilité, coût et temps. Nous écrivons critère1 $\prec$ critère2 pour indiquer que critère1 est prioritaire sur critère2 dans le choix du plan-solution.

7.1.3.1 Probabilité $\prec$ coût $\prec$ temps

Étant donné que l'utilisateur voudrait un plan avec une très grande probabilité d'exécution et afin de ne pas analyser tous les plans admissibles, nous éliminons ceux qui ont une probabilité inférieure à un seuil quelconque [Baki & Bouzid 05c, Baki & Bouzid 05a]. Déterminer un tel seuil est un problème ouvert. C'est pourquoi nous considérons que la moyenne des probabilités d'exécution des plans admissibles calculés dans la section 7.1.1 représente un seuil de comparaison des plans. Soient $\overline{Pr}$ la moyenne des probabilités d'exécution de l'ensemble $\mathcal{P}_A$ des plans admissibles et $|\mathcal{P}_A|$ sa cardinalité. La moyenne des probabilités est calculée comme suit :

$$\overline{Pr} = \frac{\sum_{a=1}^{|\mathcal{P}_A|}(Pr(\mathcal{P}_a))}{|\mathcal{P}_A|}$$

Ensuite, nous éliminons les plans admissibles qui ont une probabilité d'exécution inférieure au seuil trouvé. L'ensemble des plans vérifiant cette propriété est appelé $\mathcal{P}_V$.

Le deuxième critère de préférence par ordre de priorité est le coût. Parmi les plans sélectionnés dans l'étape précédente, nous sélectionnons celui qui a le coût total minimal. Soit $\overline{cout(t)}$ la moyenne du coût d'exécution de la tâche t. Cette moyenne est égale à la somme des coûts d'exécution des intervalles d'exécution de t divisée par leur nombre. Le coût d'exécution total d'un plan admissible $\mathcal{P}_v \in \mathcal{P}_V$ est égal à la somme des moyennes des coûts d'exécution de toutes les tâches de $\mathcal{P}_v$. Plus formellement : $cout\_total(\mathcal{P}_v) = \sum_{i=1}^{n} \overline{cout(t_i)}$ où $t_i \in T(\mathcal{P}_v)$ et n représente le nombre des tâches de $\mathcal{P}_v$. Le plan choisi est celui qui a le coût total d'exécution minimal :

$$\mathcal{P}^s = \underset{\mathcal{P}_v \in \mathcal{P}_V}{argmin} \quad cout\_total(\mathcal{P}_v) \tag{7.8}$$

Enfin, si plusieurs plans sont sélectionnés suite à l'étape précédente, le critère temps sera pris en compte. Ainsi, parmi les plans résultants, le planificateur sélectionne pour l'exécution le plan ayant le temps d'exécution minimal.

Le temps d'exécution total d'un plan admissible $\mathcal{P}_a$ est égal à la différence du maximum des dates de fin des tâches finales moins le minimum des dates de début des tâches initiales. Ce temps représente la durée de l'exécution de $\mathcal{P}_a$ entre le début au plus tôt et la fin au plus tard de son exécution effective. Plus formellement : $temps\_total(\mathcal{P}_a) = max(E_{T_F}) - min(S_{T_I})$ où T_F et T_I sont respectivement l'ensemble de tâches finales et l'ensemble de tâches initiales de $\mathcal{P}_a$. E_{T_F} et S_{T_I} représentent respectivement l'ensemble des dates de fin d'exécution possibles de T_F et l'ensemble des dates de début d'exécution possibles de T_I.

Après le calcul des temps d'exécution totaux de tous les plans-solution ($\mathcal{P}^S = \cup\mathcal{P}^s$) choisis ci-dessus, le planificateur sélectionne pour l'exécution le plan qui a le temps d'exécution total

minimal :

$$\mathcal{P}'^{s} = \underset{\mathcal{P}_s \in \mathcal{P}_S}{argmin} \quad temps\_total(\mathcal{P}^s) \tag{7.9}$$

7.1.3.2 Valeur espérée de coût/plan ≺ temps/ordonnancement

Si l'utilisateur souhaite avoir un plan-solution avec une très grande probabilité d'être exécuté ayant un coût réduit et un ordonnancement répondant à un ou plusieurs critères de préférence, dans ce cas le planificateur sélectionne le plan ayant la valeur espérée de coût totale minimale (section 7.1.2). Ensuite, il sélectionne parmi tous les ordonnancements du plan-solution celui qui répond aux attentes de l'utilisateur [Baki *et al.* 05, Baki & Bouzid 06]. Nous détaillons la partie reliée à la sélection du meilleur ordonnancement dans la section 7.2.4. En ce qui concerne la sélection du plan-solution, nous avons :

$$\mathcal{P}^s = \underset{\mathcal{P}_a \in \mathcal{P}_A}{argmin} \quad \vartheta(cout(\mathcal{P}_a)) \tag{7.10}$$

Ce plan est considéré comme le meilleur plan à exécuter en respectant toutes les contraintes et qui a une forte probabilité d'être exécuté en limitant le coût d'exécution.

Dans les sections suivantes, nous déterminons le meilleur ordonnancement des tâches du plan-solution.

7.2 Sélection du meilleur ordonnancement

Un ordonnancement d'un plan admissible $\mathcal{P}_a$ est l'ensemble des tâches de $\mathcal{P}_a$ tel que pour chacune d'entre elles on associe un de ses intervalles d'exécution possibles durant lequel elle doit s'exécuter.

L'association de chaque tâche à un de ses intervalles d'exécution doit respecter les contraintes de précédence. En effet, une tâche t_i ne peut commencer qu'après que son prédécesseur direct t_j termine son exécution - ou encore - l'intervalle d'exécution associé à la tâche t_j doit avoir une date de fin plus petite que la date de début de l'intervalle d'exécution associé à la tâche t_i. Plus formellement, soient $\mathcal{P}_a$ un plan admissible tel que $T(\mathcal{P}_a) = \{t_1, t_2, \ldots, t_n\}$ et $\mathcal{I}_{t_i} = \{I^1_{t_i}, I^2_{t_i}, \ldots, I^{m_{t_i}}_{t_i}\}$ l'ensemble d'intervalles d'exécution possibles d'une tâche $t_i \in T(\mathcal{P}_a)$. Un ordonnancement possible $\mathcal{P}_{ord} = \{(t_1, I^{k_{t_1}}_{t_1}), (t_2, I^{k_{t_2}}_{t_2}), ..., (t_n, I^{k_{t_n}}_{t_n})\}$ de $\mathcal{P}_a$, est un ensemble de couples composé chacun d'une tâche t_i appartenant à $T(\mathcal{P}_a)$ et d'un des intervalles appartenant à $\mathcal{I}_{t_i}$ vérifiant la condition : si $t_i \rightarrow t_j$ (t_i et $t_j \in \mathcal{P}_{ord}$) alors $e^{k_{t_i}}_{t_i} \leq s^{k_{t_j}}_{t_j}$ où $k_{t_i} = 1..m_{t_i}$. En fait, un plan est composé d'une combinaison de plusieurs ordonnancements c.-à-d. $\mathcal{P}_a = \bigcup \mathcal{P}_{ord}$. Nous notons $ORD(\mathcal{P}_a)$ l'ensemble des ordonnancements possibles d'un plan admissible $\mathcal{P}_a$.

Exemple 12 *Supposons que le plan admissible* $\mathcal{P}_{a_1} = [[t_2 \rightarrow t_7], [t_1 \rightarrow t_6]] \rightarrow t_{10} \rightarrow t_{15} \rightarrow t_{18}$ *est le plan-solution choisi pour l'exécution selon une des méthodes expliquées dans la première partie de ce chapitre. Les intervalles d'exécution possibles valides des tâches de* $\mathcal{P}_{a_1}$*, calculés dans la section 5.2.3 du chapitre 5, sont :*

$\mathcal{I}_{t_1} = \{[1,6]\}$	$\mathcal{I}_{t_{10}} = \{[20,24], [20,26], [20,30], [22,26], [22,28]\}$
$\mathcal{I}_{t_2} = \{[2,4], [2,5]\}$	$\mathcal{I}_{t_{15}} = \{[26,31], [26,36], [30,35], [28,33]\}$
$\mathcal{I}_{t_6} = \{[6,14], [6,15], [6,16]\}$	$\mathcal{I}_{t_{18}} = \{[31,45], [31,46], [33,47], [33,48], [35,49], [35,50], [36,50]\}$
$\mathcal{I}_{t_7} = \{[5,20], [5,22]\}$	

L'ensemble des ordonnancements possibles de $\mathcal{P}_{a_1}$ *est* $ORD(\mathcal{P}_{a_1}) = \{\mathcal{P}^1_{ord}, \mathcal{P}^2_{ord}, \ldots, \mathcal{P}^n_{ord}\}$*. Il est représenté numériquement dans la table 7.1 et graphiquement dans la figure 7.1.*

$\mathcal{P}^1_{ord} = \{[[(t_1,[1,6]) \rightarrow (t_6,[6,14])], [(t_2,[2,4]) \rightarrow (t_7,[5,20])]] \rightarrow (t_{10},[20,24]) \rightarrow (t_{15},[26,31]) \rightarrow (t_{18},[31,45])\}$
$\mathcal{P}^2_{ord} = \{[[(t_1,[1,6]) \rightarrow (t_6,[6,14])], [(t_2,[2,4]) \rightarrow (t_7,[5,20])]] \rightarrow (t_{10},[20,24]) \rightarrow (t_{15},[26,31]) \rightarrow (t_{18},[31,46])\}$
$\mathcal{P}^3_{ord} = [[(t_1,[1,6]) \rightarrow (t_6,[6,14])], [(t_2,[2,4]) \rightarrow (t_7,[5,20])]] \rightarrow (t_{10},[20,24]) \rightarrow (t_{15},[26,36]) \rightarrow (t_{18},[36,50])\}$
.....
.....
$\mathcal{P}^n_{ord} = \{[[(t_1,[1,6]) \rightarrow (t_6,[6,16])], [(t_2,[2,5]) \rightarrow (t_7,[5,22])]] \rightarrow (t_{10},[22,28]) \rightarrow (t_{15},[28,33]) \rightarrow (t_{18},[33,48])\}$

TAB. 7.1 – L'ensemble des ordonnancements possibles de $\mathcal{P}_{a_1}$

Le nombre total des ordonnancements d'un plan $\mathcal{P}_a$ noté $No(ORD(\mathcal{P}_a))$ est égal, au pire cas, au produit des cardinalités des ensembles d'intervalles d'exécution possibles de toutes les tâches de $T(\mathcal{P}_a)$ (on peut avoir un certain nombre d'intervalles ne respectant pas les contraintes de précédence entre les tâches et par conséquent l'ordonnancement correspond n'est pas valide), Plus formellement, dans le pire des cas, nous avons :

$$No(ORD(\mathcal{P}_a)) = m_{t_1} \cdot m_{t_2} \cdot \ldots \cdot m_{t_n} \tag{7.11}$$

Comme dans le cas du choix d'un plan-solution, plusieurs méthodes de choix d'un ordonnancement-solution sont possibles selon les préférences de l'utilisateur. Nous détaillons dans les sections suivantes les différentes méthodes possibles de sélection du meilleur ordonnancement (appelé ordonnancement-solution et noté $\mathcal{P}^s_{ord}$).

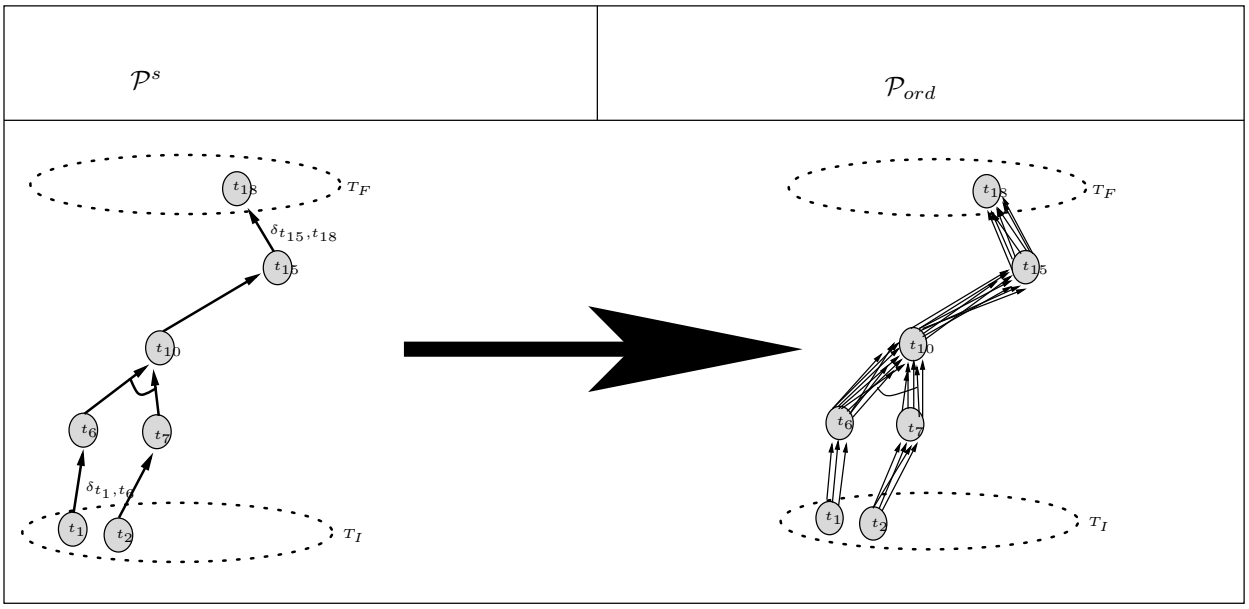

FIG. 7.1 – Les ordonnancements possibles de $\mathcal{P}^s$

7.2.1 Sélection selon la probabilité

L'ordonnancement le plus probable est déterminé en fonction des probabilités des intervalles d'exécution des tâches de celui-ci. En fait, un ordonnancement est composé d'un ensemble des tâches où à chacune est associé un seul intervalle d'exécution possible. La probabilité d'exécution d'un ordonnancement $\mathcal{P}_{ord}$ est égale au produit de toutes les probabilités des intervalles d'exécution possibles des tâches qui le composent. Plus formellement :

$$Pr(\mathcal{P}_{ord}) = \Pi_{k_{t_i}=1}^{m_{t_i}} Pr(I_{t_i}^{k_{t_i}}) \tag{7.12}$$

où $t_i \in T(\mathcal{P}_{ord})$.

L'ordonnancement le plus probable pour atteindre l'objectif, c.-à-d. celui dont la probabilité est la plus élevée, est déterminé comme suit :

$$\mathcal{P}_{ord}^s = \underset{\mathcal{P}_{ord} \in ORD(\mathcal{P}^s)}{argmax} Pr(\mathcal{P}_{ord}) \tag{7.13}$$

7.2.2 Sélection à base d'une seule fonction de valeur espérée

Comme dans le cas du choix d'un plan-solution, il est possible que l'utilisateur exprime le souhait d'avoir un compromis entre une grande probabilité d'exécution et un coût et un temps réduits. Dans ce cas il est préférable d'utiliser les valeurs espérées de coût et de temps des intervalles d'exécution des tâches. Dans cette section, nous expliquons la méthode de calcul

des valeurs espérées de coût et de temps des ordonnancements. Ensuite, nous choisissons le meilleur ordonnancement-solution en utilisant une fonction de valeur espérée valorisant l'importance qu'accorde l'utilisateur au temps et au coût.

La valeur espérée totale de coût d'un ordonnancement $\mathcal{P}_{ord}$, notée $\vartheta(cout(\mathcal{P}_{ord}))$, est calculée en additionnant toutes les valeurs espérées de coût de tous les intervalles d'exécution possibles de $\mathcal{P}_{ord}$. Plus formellement :

$$\vartheta(cout(\mathcal{P}_{ord})) = \sum_{k_{t_i}=1}^{m_{t_i}} \vartheta(cout(I_{t_i}^{k_{t_i}})) \tag{7.14}$$

Rappelons que $\vartheta(cout(I_{t_i}^{k_{t_i}})) = cout(I_{t_i}^{k_{t_i}}) \cdot Pr(I_{t_i}^{k_{t_i}})$ pour tout $t_i \in T(\mathcal{P}_{ord})$ et $k_{t_i} = 1..m_{t_i}$.

La valeur espérée totale de temps d'un ordonnancement $\mathcal{P}_{ord}$ (initialisée à $max(\vartheta(temps(I_t^k)))$ où $t \in T_I$) est calculée de deux manières différentes selon le type de la tâche courante :

- si t_i est une tâche OU et elle a un seul prédécesseur t_j ($t_j \rightarrow t_i$) avec une valeur espérée $\vartheta(temps(I_{t_j}^{k_{t_j}}))$, alors on ajoute $\vartheta(temps(I_{t_j}^{k_{t_j}}))$ à la valeur espérée totale de l'ordonnancement. Plus formellement :

$$\vartheta(temps(\mathcal{P}_{ord})) = \vartheta(temps(\mathcal{P}_{ord})) + \vartheta(temps(I_{t_j}^{k_{t_j}})) \tag{7.15}$$

- si t_i est une tâche ET et elle a un ensemble de prédécesseurs $\{t_1, t_2, \ldots, t_r\}$ ($[t_1, t_2, \ldots, t_r] \rightarrow t_i$) alors, on ajoute le maximum des valeurs espérées des intervalles de l'ordonnancement des tâches $t_1, t_2, \ldots, t_r$ à la valeur espérée totale de l'ordonnancement. Plus formellement :

$$\vartheta(temps(\mathcal{P}_{ord})) = \vartheta(temps(\mathcal{P}_{ord})) + max_{i=1}^{r}\vartheta(temps(I_{t_i}^{k_{t_i}})) \tag{7.16}$$

  tels que $\vartheta(temps(I_{t_i}^{k_{t_i}})) = temps(I_{t_i}^{k_{t_i}}) \cdot Pr(I_{t_i}^{k_{t_i}})$ pour tout $t_i \in \{t_1, t_2, \ldots, t_r\}$ et $I_{t_i}^{k_{t_i}}$ est l'intervalle d'exécution possible associé à la tâche t_i dans l'ordonnancement $\mathcal{P}_{ord}$.

Pour obtenir la valeur espérée totale d'un ordonnancement, nous additionnons ses valeurs espérées de temps et de coût. Nous pondérons cette somme par des coefficients pour permettre d'ajuster l'importance relative aux différentes valeurs espérées selon les préférences de l'utilisateur. Nous traduisons ces préférences en une fonction $\vartheta(\mathcal{P}_a)$: ϑ où ϑ est la fonction valeur espérée ayant comme coefficients α pour le coût et β pour le temps. Ces coefficients expriment l'importance qu'accorde l'utilisateur au coût ou au temps. L'équation obtenue représente un problème d'analyse multi-critères. Plus formellement, cette valeur espérée est exprimée comme suit :

$$\vartheta(\mathcal{P}_{ord}) = \alpha \cdot \vartheta(cout(\mathcal{P}_{ord})) + \beta \cdot \vartheta(temps(\mathcal{P}_{ord})) \tag{7.17}$$

tels que $\alpha + \beta = 1$, $0 \leq \alpha \leq 1$, $0 \leq \beta \leq 1$ et $\mathcal{P}_{ord} \in ORD(\mathcal{P}^s)$. Si l'utilisateur n'a pas de préférence entre le coût et le temps, il donne la même valeur à α et β (0.5). Sinon, il donne une valeur plus élevée au coefficient de la valeur de ce qu'il préfère en gardant toujours $\alpha + \beta = 1$.

Parmi tous les ordonnancements possibles du plan-solution, nous sélectionnons pour l'exécution celui qui a la valeur espérée minimale :

$$\mathcal{P}^s_{ord} = \underset{\mathcal{P}_{ord} \in ORD(\mathcal{P}^s)}{argmin} \vartheta(\mathcal{P}_{ord}) \tag{7.18}$$

D'autres méthodes de sélection de l'ordonnancement-solution basées sur le temps sont envisageables. Nous pouvons sélectionner l'ordonnancement ayant une durée minimale, une date de début au plus tôt, etc. Pour se faire, nous déterminons l'union d'intervalles pour tous les intervalles d'exécution des ordonnancements. Ensuite, nous sélectionnons l'ordonnancement répondant le mieux aux critères de préférence de l'utilisateur en terme du temps. La section suivante définie une union d'intervalles et décrit l'algorithme pour la calculer.

7.2.3 Calcul de l'union des intervalles d'un ordonnancement

Dans cette section, nous déterminons pour chaque ordonnancement, l'union des intervalles d'exécution [Bouzid 95, Baki & Bouzid 06]. Ensuite nous calculons la durée totale d'exécution relative à chaque ordonnancement. Enfin nous sélectionnons l'ordonnancement conduisant à la plus petite durée.

Exemple 13 *Afin de bien comprendre l'utilité de l'union d'intervalles, commençons par un petit exemple explicatif. Si nous devions comparer l'ordonnancement possible* $\mathcal{P}^i_{ord} = \{[[(t_1, [1, 6]) \rightarrow (t_6, [6, 14])], [(t_2, [2, 4]) \rightarrow (t_7, [5, 20])]] \rightarrow (t_{10}, [20, 24]) \rightarrow (t_{15}, [26, 31]) \rightarrow (t_{18}, [31, 45])\}$ *avec celui-ci :* $\mathcal{P}^j_{ord} = \{[[(t_1, [1, 6]) \rightarrow (t_6, [6, 16])], [(t_2, [2, 5]) \rightarrow (t_7, [5, 20])]] \rightarrow (t_{10}, [20, 26]) \rightarrow (t_{15}, [26, 31]) \rightarrow (t_{18}, [31, 46])\}$ *pour en sélectionner le meilleur, il paraît que c'est impossible puisqu'il y a des trous dans leurs intervalles d'exécution ainsi des sous intervalles communs. C'est pourquoi, il est nécessaire de les «unifier» pour les comparer. C'est pour cela que nous nous penchons sur le problème du calcul d'unions d'intervalles et ainsi rendre aisé les comparaisons entre plusieurs ensembles d'intervalles. Sur cet exemple, nous comparerons alors* $[1, 24] \cup [26, 45]$ *avec* $[1, 46]$. *Ceci nous permettra de faire des comparaisons sur les durées totales des ordonnancements, les dates de début, les dates de fin, etc. Les intervalles d'exécution possibles dans cet exemple ainsi que leurs unions d'intervalles sont représentés dans la figure 7.2.*

Nous définissons l'union des intervalles de l'ordonnancement $\mathcal{P}_{ord} = \{(t_1, I^{k_{t_1}}_{t_1}), (t_2, I^{k_{t_2}}_{t_2}),..., (t_n, I^{k_{t_n}}_{t_n})\}$ notée $Union(Int(\mathcal{P}_{ord})) = Union(I^{k_{t_1}}_{t_1}, I^{k_{t_2}}_{t_2}, \ldots, I^{k_{t_n}}_{t_n}) = \{I_1, I_2, ..., I_p\}$, par l'ensemble des intervalles obtenus en fusionnant dans un même intervalle ceux qui ont un sous intervalle

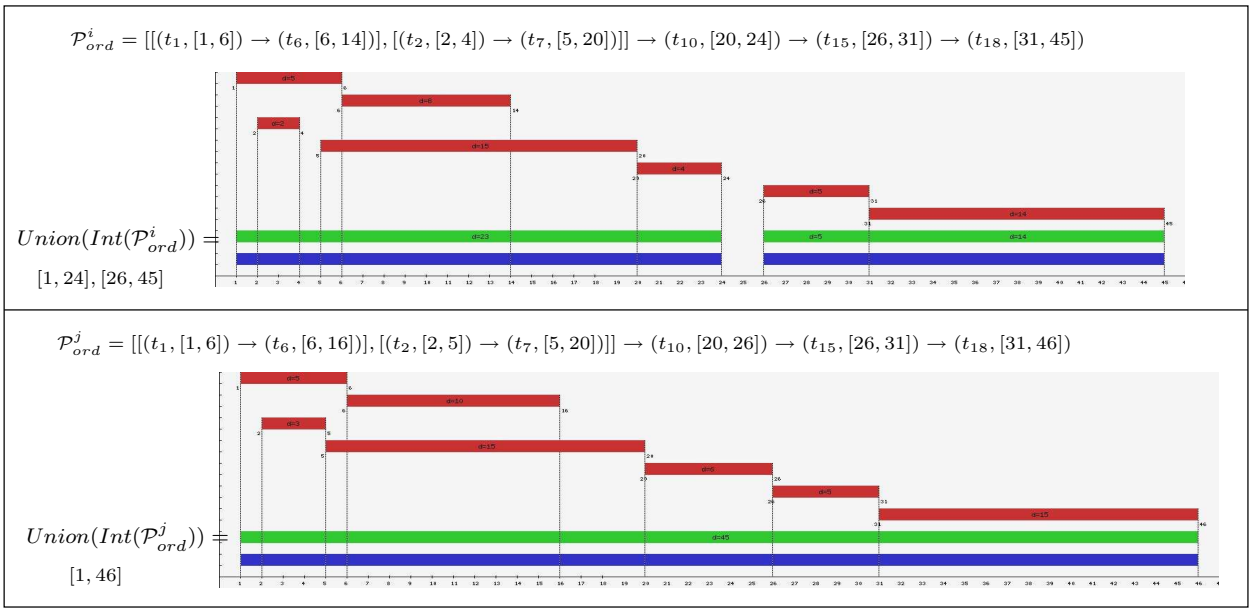

FIG. 7.2 – Les Unions d'intervalles d'exécution possibles de $\mathcal{P}^i_{ord}$ et $\mathcal{P}^j_{ord}$.

commun ou ceux qui se rencontrent. L'algorithme qui calcule l'union de l'ensemble des intervalles d'un ordonnancement, inspiré de [BOUZID 95], est décrit dans la section suivante.

7.2.3.1 Description de l'algorithme

Soit P_{Int} l'ensemble des intervalles d'exécution d'un ordonnancement possible du plan $\mathcal{P}_a$. Par exemple : $P_{Int} = \{[1,6],[2,4],[5,20],[6,14],[20,24],[26,31],[31,45]\}$. Une autre façon de voir cet ensemble est de le considérer comme un ensemble de bornes de début et de fin d'intervalles.

Afin de faire l'union des intervalles d'exécution, nous étiquetons les bornes de débuts par la lettre s et les bornes de fins par la lettre e, ensuite nous les trions dans l'ordre croissant. Nous transformons ainsi le problème de calcul d'union d'intervalles en traitement de liste ordonnée.

Exemple 14
- *Si P_{Int} est composé de deux intervalles joints* $[0,4]$ *et* $[2,6]$*, l'étiquetage et le tri des bornes de débuts et de fins donnent la liste* $[0s,2s,4e,6e]$.
- *Si P_{Int} est composé de deux intervalles disjoints :* $[0,3]$ *et* $[5,8]$*, l'étiquetage et le tri des bornes de débuts et de fins donnent la liste* $[0s,3e,5s,8e]$.
- *Si P_{Int} est composé d'un intervalle contenant totalement l'autre :* $[0,10]$ *et* $[4,8]$*, l'étiquetage et le tri des bornes de débuts et de fins donnent la liste* $[0s,4s,8e,10e]$.

Le résultat de l'algorithme est un ensemble des intervalles disjoints indiquant les différentes périodes d'exécution effectives dans un plan. L'application de l'algorithme de l'union de l'ensemble des intervalles d'un ordonnancement, représenté dans l'algorithme 18, sur l'exemple pré-

cédent donne les unions d'intervalles suivantes :

- $[0s, 2s, 4e, 6e] \rightarrow \{[0, 6]\}$.
- $[0s, 3e, 5s, 8e] \rightarrow \{[0, 3][5, 8]\}$.
- $[0s, 4s, 8e, 10e] \rightarrow \{[0, 10]\}$.

La figure 7.3 représente un exemple de trois cas de figure possibles des unions d'intervalles : les intervalles sont joints, disjoints ou les uns contients les autres. Les deux premières barres, en gris moyen, représentent les intervalles d'exécution, la troisième barre, en gris clair, représente le résultat de l'union de ces intervalles d'exécution. Remarquons que les intervalles produits d'une union d'un ensemble des intervalles d'exécution sont toujours disjoints.

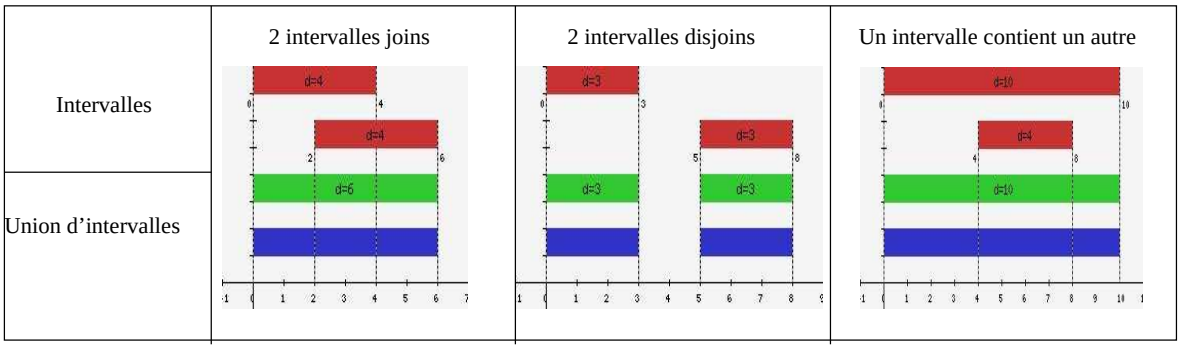

FIG. 7.3 – Union d'intervalles d'exécution

7.2.3.2 Complexité

Parce que nous ne parcourons qu'une fois pour tout le traitement de la liste de bornes, l'algorithme de calcul de l'union de l'ensemble des intervalles d'un ordonnancement a pour complexité $\mathcal{O}(n)$. La complexité du traitement global de l'ensemble des intervalles dépend de l'algorithme de tri numérique de liste ordonner().

7.2.4 Sélection à base de durée d'exécution

Soient $\mathcal{P}^s = \{t_1, t_2, ..., t_n\}$ le plan-solution sélectionné pour l'exécution et $\mathcal{P}_{ord} = \{(t_1, I_{t_1}^{k_{t_1}}), (t_2, I_{t_2}^{k_{t_2}}), ..., (t_n, I_{t_n}^{k_{t_n}})\}$ un ordonnancement possible de $\mathcal{P}^s$. Soit $Union(Int(\mathcal{P}_{ord})) = \{U_1, U_2, ..., U_p\}$ une union des intervalles de $\mathcal{P}_{ord}$ tel que $U_i = [s_i, e_i]$ est une union d'un sous ensemble des intervalles de $\mathcal{P}_{ord}$.

Nous définissons la durée totale d'exécution d'un plan $\mathcal{P}^s$ en respectant un ordonnancement comme étant la durée d'exécution nécessaire à toutes ses tâches. Celle-ci est égale à la somme des durées de tous les intervalles appartenant à l'union des intervalles de l'ordonnancement. Plus formellement, la durée effective totale d'exécution d'un ordonnancement de $\mathcal{P}^s$ est calculée

Données : FinP_{Int} : liste des bornes étiquetées s (début) ou e (fin) de l'ensemble d'intervalles traités. Les étiquettes sont concaténées à chaque borne.
ordonner() : effectue un tri numérique de liste en considérant $s < e$
premiers(ch) : retourne ch sans son dernier caractère
dernier(ch) : retourne le dernier caractère de la chaîne ch
queue(L) : retourne la liste L sans son premier élément
push(L, elt) : empile en fin de liste elt dans L

```
début
    L=ordonner(FinP_Int) ;
    co = 0 ;
    intervalle_tampon=intervalle vide ;
    array_union=liste vide ;
    tant que L! = () faire
        si (intervalle_tampon==intervalle_vide && dernier(tete(L))==«s») alors
            intervalle_tampon.borne_debut=premiers(tete(L)) ;
            co++ ;
        sinon
            si (dernier(tete(L))==«s») alors
                co++ ;
            sinon
                co−− ;
                si co == 0 alors
                    intervalle_tampon.borne_fin=premiers(tete(L)) ;
                    push(array_union,intervalle_tampon) ;
                    intervalle_tampon=intervalle vide ;
                fin
            fin
            L=queue(L) ;
        fin
    fin
fin
```

Algorithme 18 – Union de l'ensemble des intervalles d'un ordonnancement

comme suit :

$$duree\_effective(\mathcal{P}_{ord}) = \sum_{i=1}^{p}(e_i - s_i) \tag{7.19}$$

où p représente le nombre des unions d'intervalles de l'ensemble $Union(Int(\mathcal{P}_{ord}))$.

Exemple 15 *Soit* $\mathcal{P}^i_{ord} = \{[[(t_1,[1,6]) \rightarrow (t_6,[6,14])],[(t_2,[2,4]) \rightarrow (t_7,[5,20])]] \rightarrow (t_{10},[20,24]) \rightarrow (t_{15},[26,31]) \rightarrow (t_{18},[31,45])\}$ *un ordonnancement possible du plan-solution* $\mathcal{P}^s$. *L'union d'intervalles de cet ordonnancement, calculée dans l'exemple 13 en utilisant l'algorithme 18, est* $Union(Int(\mathcal{P}^i_{ord})) = \{[1,24],[26,45]\}$. *La durée effective totale d'exécution de* $\mathcal{P}^i_{ord}$ *est égale à* $(24-1)+(45-26) = 42$. *Celle de* $\mathcal{P}^j_{ord}$ *du même exemple est égale à* 45.

Cette durée exprime la durée totale effective nécessaire à l'exécution en respectant l'ordonnancement $\mathcal{P}_{ord}$. Elle est utile pour calculer des contraintes reliées à la durée d'exécution effective comme la consommation de ressources, la durée de vie des composants, etc.

L'ordonnancement-solution que nous sélectionnons est celui qui amène à la plus petite durée totale d'exécution. En d'autre terme :

$$\mathcal{P}^s_{ord} = \underset{\mathcal{P}_{ord} \in ORD(\mathcal{P}^s)}{argmin} \; duree\_effective(\mathcal{P}_{ord}) \tag{7.20}$$

Exemple 16 *Parmi les deux ordonnancements de l'exemple 15, l'ordonnancement-solution est* $\mathcal{P}^i_{ord}$ *puisqu'il a une durée totale d'exécution inférieure à* $\mathcal{P}^i_{ord}$ ($duree\_effective(\mathcal{P}^i_{ord}) = 42 <$ $duree\_effective(\mathcal{P}^j_{ord}) = 46 - 1 = 45$).

Cet ordonnancement est considéré comme le meilleur en terme de temps d'exécution effective (c.-à-d.sans le temps où l'agent ne fait rien) du plan-solution en respectant toutes les contraintes. Il s'agit d'un ensemble d'intervalles pendant lesquels il est conseillé d'exécuter les tâches pour que la durée d'exécution réelle de l'agent soit minimisée afin de réduire certaines dépenses (consommation d'énergies, de ressources, ...).

7.2.5 Autres méthodes de sélection

D'autres critères de sélection peuvent être pris en compte. En effet, un ordonnancement peut être sélectionné en fonction de sa date de début, de sa date de fin, de sa durée d'exécution, etc.

Soit $Union(Int(\mathcal{P}_{ord})) = \{U_1, U_2, ..., U_p\}$ où $U_i = [s_i, e_i]$ est une union d'intervalles de $\mathcal{P}_{ord}$, s_i et e_i étant respectivement les dates de début et de fin de l'intervalle U_i.

– **Début d'exécution au plus tôt**

L'ordonnancement qui commence le plus tôt possible est celui qui a la date de début

d'exécution la plus petite parmi toutes les dates de début de toutes les unions d'intervalles des ordonnancements possibles du plan-solution. En d'autres termes, l'ordonnancement-solution qui sera sélectionné selon ce critère est le suivant :

$$\mathcal{P}^s_{ord} = \underset{U_i = [s_i, e_i] \in Union(Int(\mathcal{P}_{ord}))}{argmin} (s_i) \tag{7.21}$$

– **Début d'exécution au plus tard**

L'ordonnancement qui commence le plus tard est celui qui a la date de début d'exécution la plus grande parmi toutes les dates de début de la première union d'intervalles des ordonnancements possibles du plan-solution. En d'autres termes, soit $U^j_1 = [s^j_1, e^j_1]$ le premier intervalle de l'ensemble d'unions d'intervalles de $\mathcal{P}^j_{ord}$ (le premier intervalle de l'ensemble d'unions d'intervalles d'un ordonnancement est celui qui a la date de début la plus petite parmi tous les intervalles d'unions du même ordonnancement) où s^j_1 et e^j_1 sont respectivement les dates de début et de fin de l'intervalle d'union U^j_1. L'ordonnancement-solution qui sera sélectionné est le suivant :

$$\mathcal{P}^s_{ord} = \underset{U^j_1 = [s^j_1, e^j_1] \in Union(Int(\mathcal{P}^j_{ord}))}{argmax} (s^j_1) \tag{7.22}$$

– **Fin d'exécution au plus tôt**

L'ordonnancement qui finit le plus tôt possible est celui qui a la date de fin d'exécution la plus petite parmi toutes les dates de fin de toutes les dernières unions d'intervalles des ordonnancements possibles du plan-solution. En d'autres termes, soit $U^j_p = [s^j_p, e^j_p]$ le dernier intervalle de l'ensemble d'unions d'intervalles de $\mathcal{P}^j_{ord}$ (le dernier intervalle de l'ensemble d'unions d'intervalles d'un ordonnancement est celui qui a la date de fin la plus grande parmi tous les intervalles d'unions du même ordonnancement) où s^j_p et e^j_p sont respectivement les dates de début et de fin de l'intervalle d'union U^j_p. L'ordonnancement-solution qui sera sélectionné est le suivant :

$$\mathcal{P}^s_{ord} = \underset{U^j_p = [s^j_p, e^j_p] \in Union(Int(\mathcal{P}^j_{ord}))}{argmin} (e^j_p) \tag{7.23}$$

– **Fin d'exécution au plus tard**

L'ordonnancement qui finit le plus tard est celui qui a la date de fin d'exécution la plus grande parmi toutes les dates de fin de toutes les unions d'intervalles des ordonnancements possibles du plan-solution. En d'autres termes,

$$\mathcal{P}^s_{ord} = \underset{U_i = [s_i, e_i] \in Union(Int(\mathcal{P}_{ord}))}{argmax} (e_i) \tag{7.24}$$

– **Durée minimale d'exécution totale**

Pour chaque ordonnancement, la durée minimale d'exécution totale (avec les durées d'attente) est égale à la différence de la date de fin d'exécution du dernier intervalle et la date du début d'exécution du premier intervalle de l'ensemble d'union d'intervalles. Plus formellement, soient $I_1^j = [s_1^j, e_1^j]$ et $I_p^j = [s_p^j, e_p^j]$ le premier et le dernier intervalle de l'ensemble d'unions d'intervalles de $\mathcal{P}_{ord}^j$, la durée minimale d'exécution de $\mathcal{P}_{ord}^j$ est :

$$duree\_totale(\mathcal{P}_{ord}^j) = e_p^j - s_1^j \tag{7.25}$$

L'ordonnancement-solution qui sera sélectionné est celui qui dure le moins en temps d'exécution total y compris les temps d'attente. En d'autres termes :

$$\mathcal{P}_{ord}^s = \underset{\mathcal{P}_{ord}^j \in ORD(\mathcal{P}^s)}{argmin} \; duree\_totale(\mathcal{P}_{ord}^j) \tag{7.26}$$

Exemple 17 *Parmi les deux ordonnancements de l'exemple 15, l'ordonnancement-solution sélectionné pour l'exécution selon les cinq méthodes présentées dans cette section est dans l'ordre :*

1. $\mathcal{P}_{ord}^i$ *ou* $\mathcal{P}_{ord}^j$ *: les deux commencent à la même date.*
2. $\mathcal{P}_{ord}^i$ *ou* $\mathcal{P}_{ord}^j$ *: les deux commencent à la même date.*
3. $\mathcal{P}_{ord}^i$ *: puisqu'il termine son exécution avant* $\mathcal{P}_{ord}^j$*.*
4. $\mathcal{P}_{ord}^j$ *: puisqu'il termine son exécution après* $\mathcal{P}_{ord}^i$*.*
5. $\mathcal{P}_{ord}^i$ *: puisque* $duree\_totale(\mathcal{P}_{ord}^i) = 44$ *est inférieure à* $duree\_totale(\mathcal{P}_{ord}^j) = 45$*.*

7.3 Conclusion

L'introduction du temps, du coût et de probabilité dans un même système de planification complique la recherche d'un seul plan respectant toutes les contraintes de l'environnement et ayant un temps et un coût réduits tout en maximisant la probabilité d'exécution.

Dans ce chapitre, nous avons traité ce problème par des techniques de sélection multi-critères du plan-solution et ensuite d'un ordonnancement-solution. Nous avons utilisé les valeurs espérées des tâches calculées dans le chapitre précédent afin de calculer les valeurs espérées totales des plans admissibles pour en choisir le meilleur. Nous avons proposé plusieurs méthodes de sélection du meilleur plan-solution et ordonnancement-solution dont ceux les plus probables, ceux qui sont un compromis entre le temps/coût minimaux et la probabilité maximale, ceux qui durent le moins, etc.

Chapitre 8

Exemple : gestion d'une situation de crise

La méthode de planification que nous avons développée dans les chapitres précédents s'applique à un seul agent ayant un ensemble de tâches à exécuter pour satisfaire un ou plusieurs buts tout en respectant les contraintes de l'environnement. Mais en pratique, plusieurs situations demandent la coopération de plusieurs agents pour résoudre un problème. Dans le cas d'un tremblement de terre, d'un incendie ou d'une inondation par exemple, le secours des victimes est plus rapide, plus efficace et plus important si un ensemble de personnes (ou d'agents) travaillent ensemble. Mais pour que leur travail soit efficace, il faut qu'il y ait une bonne coordination et une bonne coopération entre eux, d'autant plus que dans la plupart des cas, une situation de crise[41] est très contrainte en temps, ressources et incertitude. Dans ce chapitre, nous allons présenter l'application de notre méthode de planification à un ensemble d'agents afin qu'ils réagissent ensemble pour atteindre les buts visés dans le cas d'une situation de crise.

Nous allons tout d'abord introduire *"Robocup Rescue"*, une compétition internationale dans laquelle des Systèmes Multi-Agents (SMA) sont évalués selon leur capacité à secourir des civils pris dans un tremblement de terre ; Ensuite, nous décrirons l'application de notre système de planification sur des situations de crise comme celles que nous pouvons trouver dans les problèmes traités dans *Robocup Rescue*. Notons que cette méthode ne traite pas le problème dans sa globalité mais propose une solution partielle au problème.

[41] Une situation de crise provient d'un changement brutal dans une organisation ou dans un état. Cela correspond par exemple à des catastrophes naturelles : tremblements de terre, inondations, ouragans, etc. La situation de crise correspond alors à la période s'étendant du début de la crise à un laps de temps variable, selon sa nature. Dans le cas d'un tremblement de terre, elle s'étend du début de la première secousse à 72 heures après la fin de celle-ci, c'est-à-dire le temps durant lequel des équipes de secours pourront retrouver des survivants dans les zones sinistrées.

8.1 Introduction à la Robocup Rescue

La *Robocup Rescue* [Robocup Rescue, Salles 04] est une compétition internationale qui concerne le développement des agents représentant des équipes de secours intervenant après un tremblement de terre ayant touché une grande ville. Il existe deux ligues : "robotique" et "simulation". Dans la première, des robots doivent explorer les décombres pour y retrouver les blessés. Dans la seconde, des équipes de secours virtuels doivent s'organiser pour secourir des civils dans le tremblement de terre. La ville simulée se veut la plus réaliste possible, avec des routes impraticables, des départs de feu, etc.

Le simulateur de la *Robocup Rescue* utilise la carte de la ville de Kobe au Japon où nous pouvons trouver les routes, les bâtiments, les incendies, les civils et les agents de secours. Une simulation comporte 300 pas de temps discret. Chaque pas de temps est un cycle qui correspond à une minute en temps réel. Le but de la compétition est de maximiser un score calculé en fonction du nombre de civils encore vivants et de leur état de santé, ainsi que de l'importance des dégâts matériels : bâtiments effondrés, dégâts dûs aux incendies, etc.

8.2 Gestion d'un exemple de situation de crise

Supposons que dans une ville, comme dans la *Robocup Rescue*, plusieurs agents hétérogènes répartis géographiquement sur plusieurs lieux aient la responsabilité de résoudre certains problèmes dans des domaines bien déterminés. Imaginons, par exemple, qu'un feu se déclenche dans une maison dans la ville. Les agents doivent réagir le plus vite et le plus efficacement possibles pour éteindre le feu. De ce simple exemple, nous pouvons retirer les principales caractéristiques de notre système. Une ville est représentée sous la forme d'une carte, où les carrés désignent les différents lieux (maisons, bâtiments, centres commerciaux,...) et les lignes qui les relient sont les chemins possibles entre eux. Pour aller d'un lieu à un autre, il faut emprunter le chemin entre eux. Parfois, il est nécessaire de passer par d'autres lieux intermédiaires afin d'accéder au lieu destination. Les agents sont repartis sur les lieux et partagent certaines caractéristiques comme la capacité de se déplacer, les modes de recevoir les ordres, etc. Ils ont néanmoins des caractéristiques différentes d'un agent à un autre comme le type de l'agent (ce qu'il peut faire), son état (disponible ou non), etc. Dans l'exemple de la figure 8.1, nous distinguons deux sortes d'agents :

1. les *unités de police (PU)*[42] : pour organiser les accès routiers,
2. les *unités de pompiers (FB)*[43] : pour éteindre les feux.

Nous supposons l'existence de deux postes de police (PP_1 et PP_2) localisées respectivement dans L_0 et L_4 et d'une caserne de pompiers (CP) localisée dans L_0. Un poste de police contient

[42] Police Units
[43] Fire Brigade

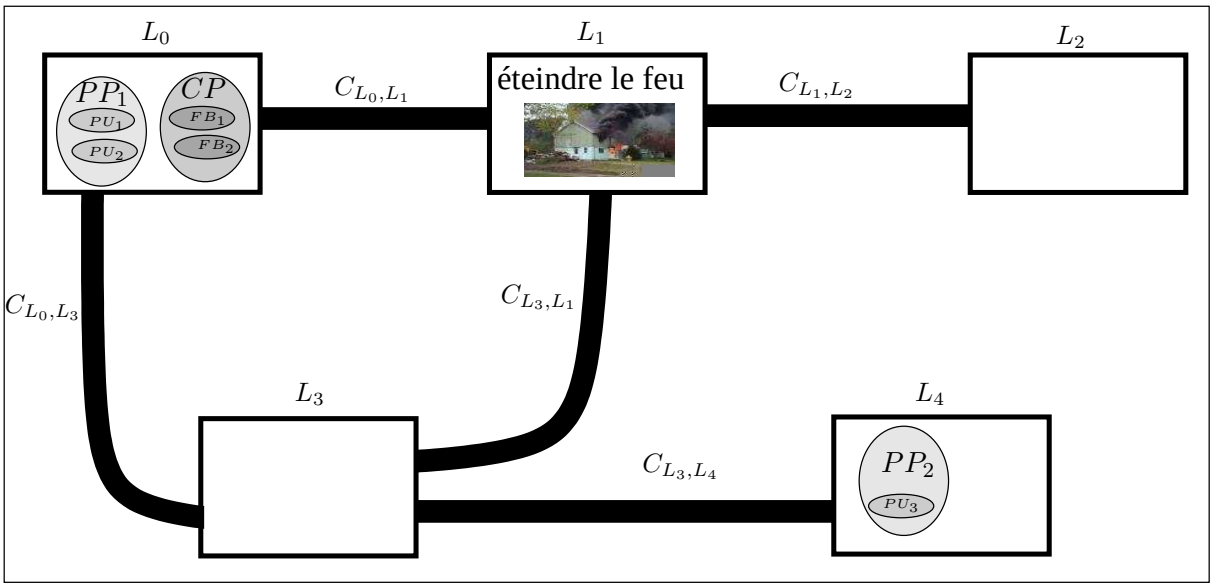

FIG. 8.1 – Exemple d'une carte avec une mission à résoudre

plusieurs unités de police, chacune est composée d'un ensemble de policiers. Une caserne de pompiers contient plusieurs unités de pompiers, chacune est composée d'un ensemble de pompiers. Toutes les unités de même type sont capables de faire exactement le même travail. Autrement dit, il n'y a aucune différence entre le travail d'une unité de police appartenant au poste PP_1 et d'une autre appartenant au poste PP_2. Dans cet exemple, nous considérons qu'une unité est un agent, c.-à-d. que nous ne cherchons pas un policier ou un pompier mais une unité de police ou une unité de pompiers.

La question à résoudre est la suivante : quelle est la meilleure manière d'atteindre l'objectif *"éteindre le feu"* dans L_1, sachant que pour ce faire, on a besoin d'une unité de pompiers et d'une unité de police ?

Supposons que PP_1 est composé de deux unités de police (PU_1 et PU_2), que PP_2 est composé d'une seule unité de police (PU_3) et que CP est composée de deux unités de pompiers (FB_1 et FB_2).

La réponse à cette question n'est pas simple, puisque plusieurs solutions sont possibles. En effet, pour *éteindre le feu* de l'exemple de la figure 8.1, nous pouvons faire appel à PU_1, PU_2 ou PU_3 et à FB_1 ou FB_2. Les combinaisons possibles de ces choix forment l'ensemble des solutions possibles :

- PU_1 et FB_1
- PU_1 et FB_2
- PU_2 et FB_1
- PU_2 et FB_2

Environnement	incertain
Structure de système	planification centralisée et hors-ligne
Méthodes de recherche	graphe ET/OU et recherche en chaînage arrière
Contraintes	temporelles, agent(type,état,...), but(type,coût,temps,probabilité)
Buts	coût, temps, probabilité, localisation
Agents	hétérogènes

TAB. 8.1 – Les caractéristiques de notre système

- PU_3 et FB_1
- PU_3 et FB_2

De plus, chacune de ces propositions peut résoudre l'objectif de plusieurs façons. Par exemple, pour *éteindre le feu* en utilisant PU_1 et FB_1, les solutions possibles sont :

1. PU_1 se déplace de L_0 à L_1 et FB_1 se déplace de L_0 à L_1.
2. PU_1 se déplace de L_0 à L_1 et FB_1 se déplace de L_0 à L_3 ensuite de L_3 à L_1.
3. PU_1 se déplace de L_0 à L_3 ensuite de L_3 à L_1 et FB_1 se déplace de L_0 à L_1.
4. PU_1 se déplace de L_0 à L_3 ensuite de L_3 à L_1 et FB_1 se déplace de L_0 à L_3 ensuite de L_3 à L_1.

Pour résoudre ce petit exemple (5 lieux, 5 chemins, 5 agents, 2 types d'agents et 1 seul objectif), nous avons 24 solutions possibles.

Maintenant, pour choisir une seule solution (la meilleure), il faut analyser toutes les propositions possibles en utilisant un ensemble de données (probabilité, coût, distance, vitesse, etc.) et de contraintes (temps, occupation, etc.).

Un agent peut être disponible ou non, il peut emprunter un chemin pendant une de plusieurs durées possibles avec une certaine probabilité et un certain coût. Sur certains chemins, il doit respecter des contraintes temporelles : par exemple, il ne peut pas utiliser un chemin avant une certaine date s'il est barré par des travaux et il doit sortir d'un chemin avant une certaine date puisqu'il sera bloqué, etc. La durée de traversé d'un chemin dépend de la qualité du chemin et de sa longueur, du véhicule utilisé et de sa vitesse, des embouteillages, etc. Nous supposons que ces informations sont déjà données et enregistrées dans une base de connaissances. Les caractéristiques de notre système sont représentées dans la table 8.1.

À un instant donné, un agent peut avoir un de cinq états différents :(1) il est en train d'exécuter une tâche, (2) il vient de terminer l'exécution d'une tâche et commencer l'exécution d'une autre (3) il est prêt pour commencer l'exécution des tâches qui lui sont déjà attribuées, (4) il est disponible ou (5) il est hors-service pour une raison ou une autre.

Dans le cas (4), l'agent est disponible et prêt à se voir donner des tâches pour les exécuter ; dans tous les autres cas, il est non disponible (soit il est occupé, soit il a déjà des tâches à

exécuter, soit il ne peut rien faire).

8.2.1 Description du domaine

Les agents doivent collaborer pour exécuter au mieux le travail. Cette collaboration se fait via un agent central dont le rôle est d'organiser le travail de tous les autres agents. Dans cette section, nous définissons les agents, la carte, la tâche et l'action.

8.2.1.1 Agent

Un agent, comme défini dans la section 5.1.1 du chapitre 5, doit exercer un processus cognitif lui permettant d'analyser l'environnement et de déterminer les meilleurs actions à entreprendre. Dans ce chapitre, nous distinguons deux sortes d'agents : agent central et agent exécutif.

1. L'agent central est celui qui s'occupe de la planification ; il a une base de connaissances contenant les informations à propos de l'environnement : la carte de la ville, les agents, les buts, les contraintes, etc. C'est le planificateur présenté dans les chapitres précédents.
2. Tous les autres agents reçoivent les commandes de la part de l'agent central et les exécutent. Ils ne peuvent pas communiquer entre eux. Le plan à exécuter doit être communiqué aux agents avant le début de l'exécution réelle des tâches. Ces agents sont regroupés en plusieurs sous-ensembles d'agents spécialisés dans des domaines bien déterminés. Les agents du même groupe sont dits des agents homogènes et sont capable de réaliser le même travail. Les agents appartenant à des groupes différents sont dits hétérogènes et réalisent des tâches de différents types.

   Chaque agent est défini par son nom et sa spécialité. La spécialité décrit ce que l'agent est capable de faire. Soit $A = \{a_1, a_2, \ldots, a_n\}$ un ensemble d'agents travaillant dans une ville représentée par une carte. Chaque agent a_i a une seule spécialité notée $Spec(a_i)$.

Exemple 18 *Dans l'exemple de la figure 8.1, nous avons deux groupes d'agents autre que l'agent central : les unités de police (PU) et les unités de pompiers (FB).* $A = \{PU_1, PU_2, PU_3, FB_1, FB_2\}$ *est l'ensemble de tous les agents tels que* $Spec(PU_1) = Spec(PU_2) = Spec(PU_3) = police$ *et* $Spec(FB_1) = Spec(FB_2) = pompier$. *Par exemple,* PU_1 *et* FB_1 *sont hétérogènes, mais* PU_1 *et* PU_2 *sont homogènes.*

À un instant t, un agent a est dans un état $s_t(a)$ définissant en plus de son nom et sa spécialité, son état de disponibilité et sa localisation sur la carte. $s_t(a)$ est décrit par la liste $\langle nom, spécialité, disponibilité, localisation\rangle$ où :

- *nom* est l'identification de l'agent (PU_1),
- *spécialité* décrit sa spécialité ($Spec(PU_3) = police$),
- *disponibilité* décrit si l'agent est disponible ou non ($PU_1 : dispo$),

– *localisation* décrit l'adresse de l'agent sur la carte ($localisation(PU_3) = L_4$).

Exemple 19 *Par exemple, dans la figure 8.1, l'état de l'agent* PU_1 *à l'instant* 0 *est le suivant :* $s_0(PU_1) = \langle PU_1, police, dispo, L_0 \rangle$.

8.2.1.2 Carte

Tous les agents sont repartis sur une carte qui définit une zone géographique comme une ville, un centre commercial, un hôpital, etc. Dans une carte, il y a deux identités importantes : les lieux et les chemins. Un lieu L est représentée par un carré et un chemin C_{L_i,L_j} entre deux lieux L_i et L_j est représenté par une ligne entre ces derniers. Les agents sont situés sur les lieux et se déplacent en empruntant les chemins.

Nous supposons que touts les chemins sont à double sens, c.-à-d. qu'un chemin qui va de L_i à L_j permet aussi d'aller de L_j à L_i.

Exemple 20 *Dans la carte de la figure 8.1, l'ensemble des lieux est* $L = \{L_0, L_1, L_2, L_3, L_4\}$ *et l'ensemble des chemins est* $C = \{C_{L_0,L_1}, C_{L_1,L_2}, C_{L_0,L_3}, C_{L_1,L_3}, C_{L_3,L_4}\}$.

Le temps de déplacement d'un lieu à un autre dépend de la distance entre eux, de l'état du chemin, de la vitesse de l'agent et d'autres paramètres.

8.2.1.3 Tâche et action

Afin d'atteindre les buts dans la carte, les agents doivent exécuter des tâches en effectuant des actions. Rappelons qu'une tâche (définie dans la section 5.1.2 du chapitre 5) possède une date de début au plus tôt, une date de fin au plus tard, un ensemble de durées d'exécution possibles ; à chacune de ces durées sont associés une probabilité et un coût d'exécution.

Définition 5 *Soit* S *l'ensemble des états possibles d'un agent. Une action est une fonction* $ac : S \rightarrow S$ *qui remplace les valeurs des paramètres de l'état actuel d'un agent par d'autres valeurs. Un agent exécute les tâches en utilisant les actions. Seulement les deux paramètres "disponibilité" et "localisation" d'un état d'un agent changent suite à l'exécution des actions.*

Exemple 21 *Soit* $s_0(PU_1) = \langle PU_1, police, dispo, L_0 \rangle$ *l'état de* PU_1 *à l'instant* 0.
$aller(PU_1, L_0, L_1, s_t)$ *est une action qui remplace la valeur* L_0 *du paramètre "localisation" de l'état de l'agent* PU_1 *par la valeur* L_1 *à l'instant* t. *Après l'exécution de cette action, l'état de l'agent* PU_1 *devient* $s_t(PU_1) = \langle PU_1, police, dispo, L_1 \rangle$.

Le but est une tâche qui possède en plus des données spéciales de tâches, la localisation du but et les spécialités des agents demandés pour sa réalisation.

Exemple 22 *eteindre_le_feu* $= \langle L_1, \{police, pompier\}, [0, 110], \{60, 90\}, \{0.75, 0.25\}, \{50, 100\}\rangle$ *est une tâche "but". Les actions nécessaires pour exécuter cette tâche sont : "verser de l'eau" pour les pompiers et "organiser les accès routiers" pour les policiers.*

8.2.2 Planification d'agents

La planification que nous proposons est une planification hors-ligne faite par l'agent central qui a une vue globale sur l'environnement. Dans cette section, nous définissons tout d'abord le domaine de planification d'agents centralisée, le problème de planification d'agents et la solution de ce problème. Cette solution se déroule en trois étapes :

1. Transformation du problème de planification d'agents en graphe ET/OU.
2. Génération d'un plan-solution.
3. Transcription du plan-solution en commandes.

Définition 6 *Un domaine de planification d'agents DPA est définie par la liste* $\langle M, S, Ac\rangle$ *où :*
- M *est la carte de localisation,*
- S *est l'ensemble des états des agents dans* M*,*
- Ac *est un ensemble d'actions comme décrit dans la définition 5.*

Définition 7 *Soit* $DPA = \langle M, S, Ac\rangle$ *un domaine de planification d'agents. Un problème de planification d'agents PPA de DPA est une liste* $\langle DPA, B, C\rangle$ *où* B *est un ensemble d'états finals (buts) et* C *est un ensemble de contraintes.*

Exemple 23 *Dans la figure 8.1,* M *est représentée par les lieux et les chemins,* $S = \{\langle PU_1,$ *police*, *dispo*, $L_0\rangle$, $\langle PU_2$, *police*, *dispo*, $L_0\rangle$, $\langle PU_3$, *police*, *dispo*, $L_4\rangle$, $\langle FB_1$, *pompier*, *dispo*, $L_0\rangle$, $\langle FB_2$, *pompier*, *dispo*, $L_0\rangle\}$, $B = \{\langle$*eteindre_le_feu*, L_1, $\{police, pompier\}$, $[0, 110]$, $\{60, 90\}$, $\{0.75, 0.25\}$, $\{50, 100\}\rangle\}$ *et* $C = \{I_g^- = 0, I_g^+ = 110\}$.

Définition 8 *Soient* $DPA = \langle M, S, Ac\rangle$ *un domaine de planification d'agents et* $PPA = \langle DPA, B, C\rangle$ *un problème de planification d'agents PPA de DPA.* Π *est une solution de PPA si et seulement si tous les états de* B *sont atteints en respectant toutes les contraintes du domaine.*

La résolution d'un PPA consiste à trouver une suite d'actions à réaliser par un sous-ensemble d'agents à des dates précises de telle façon que l'ensemble de buts soit atteint tout en respectant toutes les contraintes du domaine. Pour ce faire, (1) nous transformons tout d'abord la carte avec toutes ses données et ses contraintes en un graphe ET/OU (définition 1 du chapitre 5). Ensuite, (2) nous sélectionnons le plan-solution selon une des méthodes décrites dans le chapitre 7. Enfin, (3) nous traduisons le plan-solution trouvé sous la forme de commandes compréhensibles par les agents. Tout ce travail se fait par l'agent central qui communique ensuite le résultat

obtenu aux agents concernés par l'exécution. Ceux-ci n'ont aucune sorte de communication entre eux, par contre ils peuvent communiquer avec l'agent central pour lui donner l'état actuel de l'exécution. Ce point est une idée de poursuite de cette thèse qui se concentre sur la planification et l'ordonnancement et ne traite pas l'étape d'exécution des tâches.

Dans ce qui suit, nous présentons les trois étapes principales de la résolution d'un PPA.

8.2.2.1 Transformation d'un PPA en graphe ET/OU

La première étape de la résolution d'un problème de planification d'agents consiste à le transformer en graphe ET/OU (figure 8.2). Les nœuds d'un tel graphe représentent des tâches de type *ET* et *OU* et les arcs les relations de précédence entre elles. Nous expliquons les démarches de la transformation de la carte en graphe ET/OU par l'intermédiaire de l'exemple de la figure 8.1. Pour construire le graphe ET/OU, le planificateur commence par le lieu qui contient le but à satisfaire et construit un nœud pour être la racine du graphe. Pour atteindre ce but, on a besoin des agents dont la spécialité est du même type que le but. C'est pourquoi le planificateur construit des nœuds appelés *"nœuds artificiels"* et les relie avec la racine par des arcs orientés de ces nœuds vers la racine. Le nombre de nœuds (et par suite d'arcs) est égal au nombre d'agents demandés pour la résolution du but. Ces nœuds ont pour rôle la propagation des besoins pour atteindre le but.

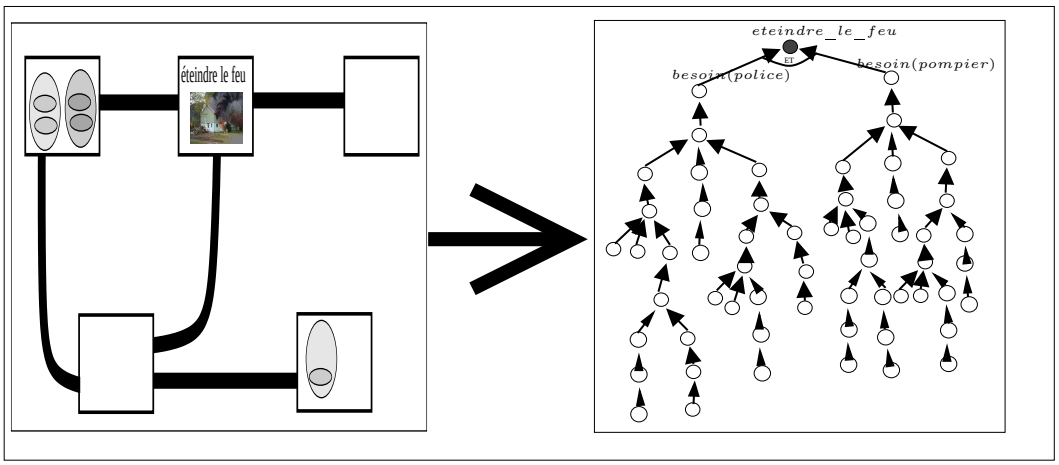

FIG. 8.2 – La première étape de la résolution d'un PPA : transformation d'un PPA en graphe ET/OU

Exemple 24 *Pour "eteindre_le_feu" dans le lieu L_1, on a besoin d'un agent de type "police" et d'un autre de type "pompier". Le planificateur construit tout d'abord la racine du graphe représentée par un nœud contenant l'information "eteindre_le_feu". Comme on a besoin de deux agents, il construit deux nœuds et deux arcs entre eux et la racine. Le premier nœud contient*

l'information "besoin(police)" et le deuxième nœud contient l'information "besoin(pompier)". Puisque la réalisation du but a besoin à la fois d'un agent police et d'un agent pompier, le nœud "eteindre_le_feu" est une tâche de type ET. Cette partie du graphe est représentée dans la première ligne de la figure 8.3.

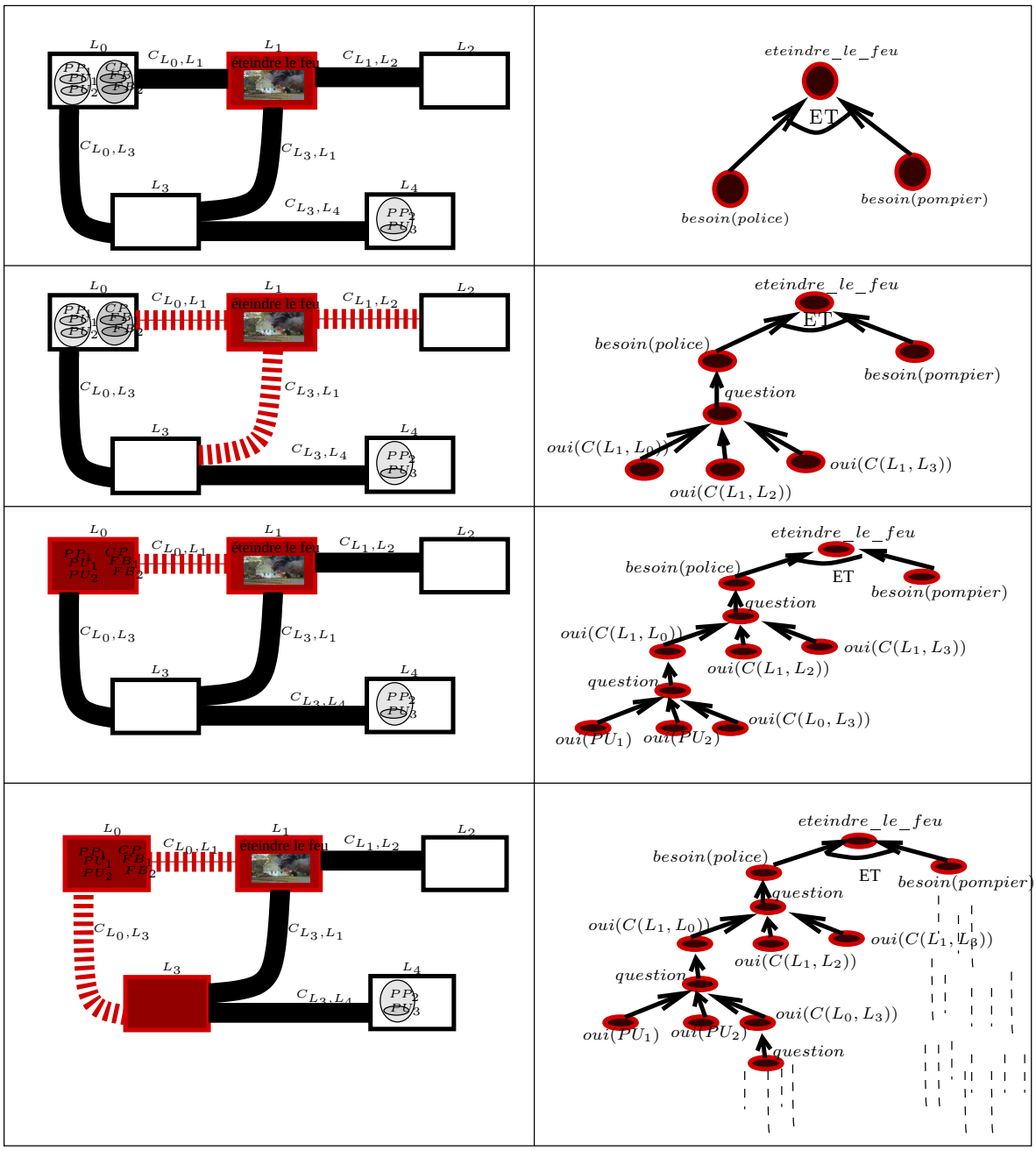

FIG. 8.3 – L'application de l'algorithme 19 sur l'exemple de la figure 8.1

Pour résoudre un nœud "$besoin(Spec(agent))$", le planificateur demande si dans le lieu où il est, il y a un agent disponible de cette spécialité ou s'il y a des chemins entre ce lieu et d'autre lieu sans qu'il repasse par des lieux déjà explorés. Cette question (représentée par un nœud appelé "*question*") est composée de deux parties :

1. $existe\_agent(Spec(agent))$ et $dispo\_agent(Spec(agent))$
2. $existe\_chemin(L_{courante}, ?)$ et $non\_cycle()$

S'il trouve un agent disponible de la spécialité recherchée, il construit un nœud *"oui(agent)"* et le relie par un arc orienté avec le nœud "*question*" ci-dessus. Ensuite il répond à la deuxième partie de la question. Sinon il répond tout simplement à la deuxième partie de la question. S'il trouve un chemin reliant le lieu courant à d'autres lieux sans qu'il repasse par un lieu déjà parcouru (c.-à-d. que la réponse à la $2^{\text{ème}}$ partie est positive), il construit un nœud pour chaque nouveau lieu trouvé et il repose la question avec les nouvelles données. Si la réponse aux deux parties de la question est négative, il construit un nœud *"non"* et le relie par un arc orienté avec le nœud "*question*" en dessus. Le planificateur arrête le développement d'une branche du graphe quand il rencontre un nœud *"oui(agent)"* ou un nœud *"non"* qui représentent les feuilles du graphe. Les étapes de construction d'un graphe ET/OU à partir d'une carte sont représentées dans la figure 8.3.

Exemple 25 *Le planificateur relie par un arc orienté vers le nœud "besoin(police)" un nœud "question" qui contient l'information :*

$[existe\_agent(police)$ *et* $dispo\_agent(police)]$ *ou* $[existe\_chemin(L_1, ?)$ *et* $non\_cycle()]$.

Ensuite, il répond aux deux parties de cette question. La réponse à la première partie est négative puisqu'il n'y a pas un agent de spécialité "police" dans le lieu L_1*. La réponse à la deuxième partie est positive, donc le planificateur construit trois nœuds* $oui(C(L_1, L_0))$, $oui(C(L_1, L_2))$ *et* $oui(C(L_1, L_3))$ *avec des arcs orientés vers le nœud "question" (*$2^{\text{ème}}$ *ligne de la figure 8.3). Respectivement et à chacun de ces nœuds, il repose la question, avec les données actuelles, représentée par un nœud "question". La réponse à la première partie de la question* $[existe\_agent(police)$ *et* $dispo\_agent(police)]$ *ou* $[existe\_chemin(L_0, ?)$ *et* $non\_cycle()]$ *est positive dans le lieu* L_0*. C'est pourquoi le planificateur construit un nœud feuille "oui(agent)" et continue la construction du graphe avec la deuxième partie de la question et les autres nœuds "question" (*$3^{\text{ème}}$ *ligne de la figure 8.3). Le planificateur continue de la même façon (ajout des nœuds et des arcs) jusqu'à ce qu'il ne reste aucun nœud non exploré. À noter que toutes les feuilles de ce graphe sont des nœuds "oui(agent)" ou "non".*

Le graphe obtenu (figure 8.4) est un type particulier du graphe ET/OU. Plus précisément :

- Les nœuds ne représentent pas nécessairement des tâches à exécuter et ils sont de plusieurs sortes :

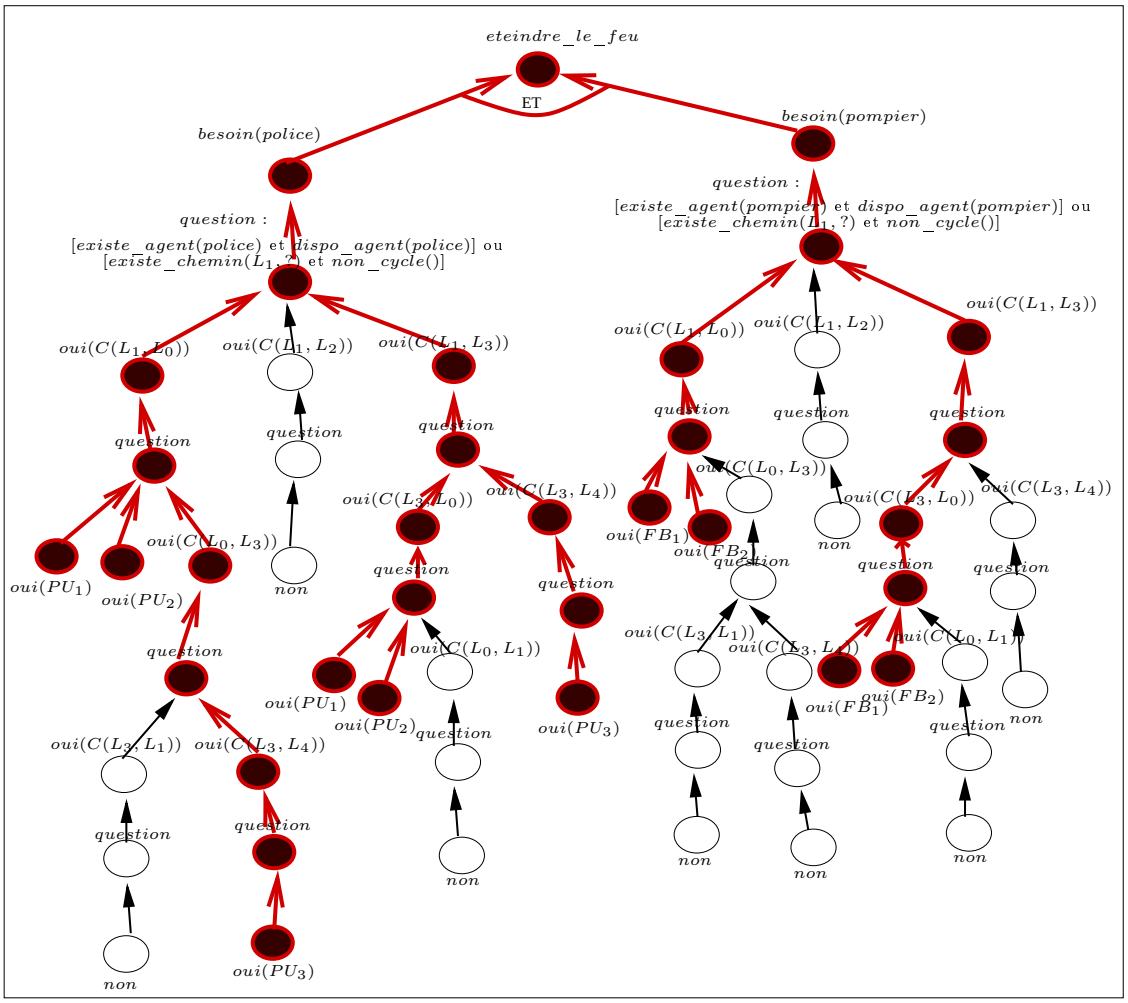

FIG. 8.4 – Le graphe ET/OU obtenu de la carte de la figure 8.1

1. Les nœuds artificiels ont pour but de propager les différentes possibilités. Ces nœuds qui ont la liste des données suivantes : $\langle I^- = 0, I^+ = \infty, \Delta = \emptyset, Pr = \emptyset, C = \emptyset \rangle$ sont : *"besoin(Spec(agent))"*, *"question"*, *"oui(agent)"* et *"non"*.
2. Les nœuds exécutables représentent les actions et les tâches (comme "$eteindre\_le\_feu$").
3. Les nœuds qui représentent l'existence d'un chemin entre deux lieux ($oui(C(L_i, L_j))$) vont être transformer en nœuds exécutables dans une étape suivante.

– Il n'y a pas une représentation des délais entre les tâches.

– Les données quantitatives de la carte ne sont pas représentées dans les nœuds.

L'algorithme de transformation d'une carte en un graphe ET/OU est représenté dans l'algorithme 19 et résumé comme suit :

– On part d'un but et on cherche les agents nécessaires à sa résolution.

– On parcourt tous les chemins qui partent du lieu du but vers les autres lieux. Dans chaque

lieu d'arrivée on vérifie s'il y a un agent disponible de la spécialité cherchée et si on est revenu à un lieu déjà exploré.

– On répète la recherche jusqu'à ce qu'on explore tous les lieux de la carte.

Données : Carte géographique (lieux, chemins, buts, agents)
Résultat : Graphe ET/OU
début
  Graphe ← ∅
  Ajouter le ou les buts en tête du *Graphe* (nœuds buts)
  Ajouter les agents nécessaires à la résolution du problème (nœuds besoins)
  pour chaque *nœud besoin* **faire**
    lieuCourant ← localisation but
    Rechercher(lieuCourant)
  fin
  pour chaque *chemins* **faire**
    lieuCourant ← lieu d'arrivée
    Rechercher (lieuCourant)
  fin
fin

Algorithme 19 – Transformation d'une carte à un graphe ET/OU

8.2.2.2 Application du planificateur

Étant obtenu un type particulier du graphe ET/OU $G = (T, E, D)$ tels que :

– $T = T_I \cup T_M \cup T_F$ où T_I contient les nœuds *"oui(agent)"* et les nœuds *"non"*, T_F contient les nœuds *"buts"* et T_M contient les autres nœuds (*besoin, question, oui*$(C(L_i, L_j))$). Rappelons que ces nœuds ne sont pas nécessairement des tâches exécutables.

– E est l'ensemble des contraintes entre les nœuds (représentées par les arcs orientés),

– $D = \emptyset$,

l'étape suivante consiste à appliquer la méthode de planification que nous avons proposée dans les chapitres précédents avec quelques petites modifications, à savoir : (1) le planificateur détermine l'ensemble des *plans faisables*, (2) élimine tous les nœuds artificiels pour obtenir l'ensemble des *plans faisables exécutables*, (3) transforme les nœuds de chaque *plan faisable exécutable* en une action à faire ou une tâche à exécuter pour obtenir un ensemble des *plans temporels faisables exécutables*. Par affinements successifs, (4) il sélectionne parmi eux les *plans admissibles*. Parmi ces plans, (5) il choisit, selon un ou plusieurs critères de préférence, le meilleur plan-solution à exécuter et enfin le meilleur ordonnancement-solution. La figure 8.5 résume ces étapes de planification.

```
Fonction Rechercher(lieu)
début
    tant que lieu possède chemin inexploré faire
        Mémoriser chemin comme emprunté
        Ajouter chemin au Graphe
        si agent localisé et disponible alors
            Marquer sa localisation
            Chemin suivant
        sinon
            Chemin suivant
        fin
    fin
fin
```

Algorithme 20 – La fonction Rechercher de l'algorithme 19

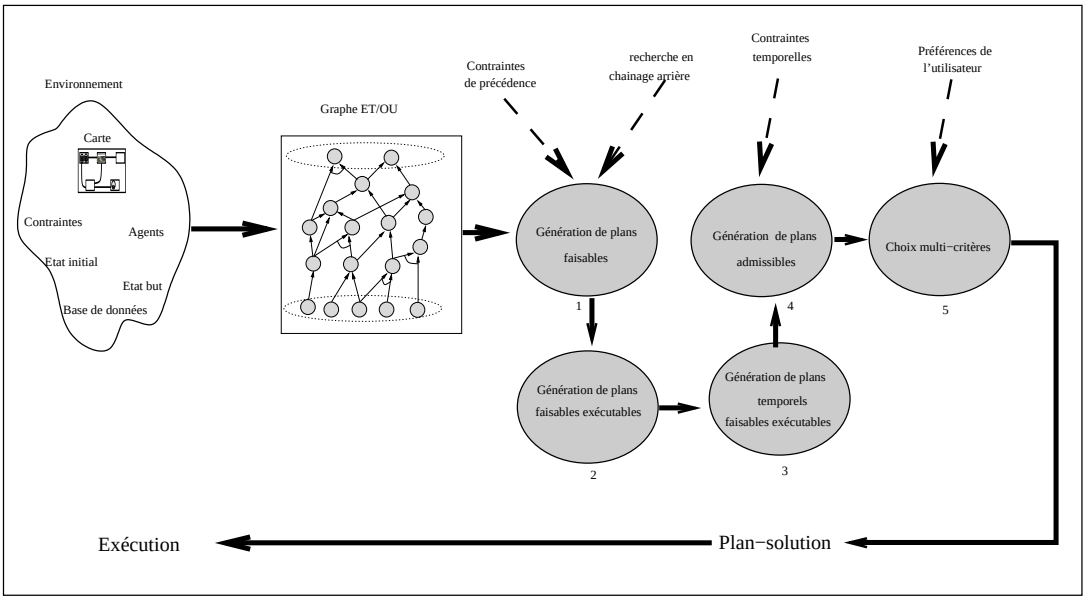

FIG. 8.5 – Les étapes de planification pour résoudre un *PPA*

Notons que les étapes (1), (4) et (5) sont les trois étapes de planification présentées dans la section 5.2 du chapitre 5. La figure 8.6 représente les étapes de planification selon la méthode présentée dans les chapitres précédents et celles de la méthode appliquée dans ce chapitre.

1. Tout d'abord, il génère les **plans faisables** du graphe ET/OU entre les nœuds initiaux (*"oui(agent)"*) et les nœuds *"buts"*. Ces plans sont des sous-graphes du graphe ET/OU. La méthode de recherche utilisée est celle de chaînage en arrière. Pour les détails revoir la

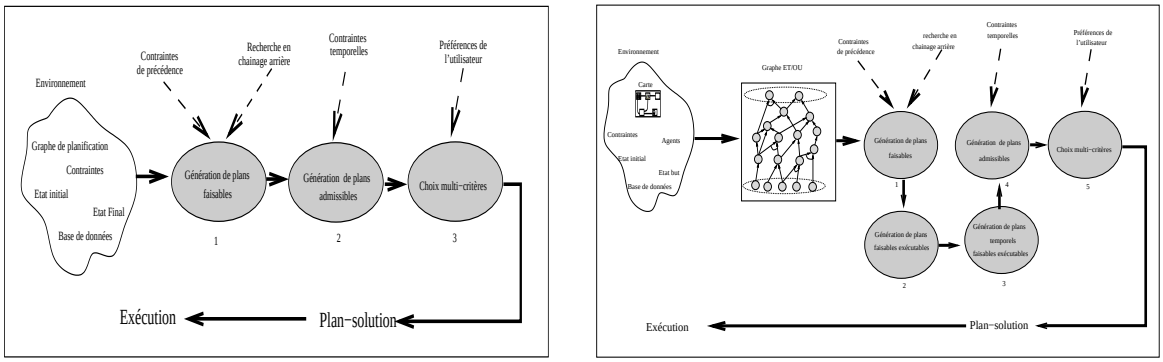

FIG. 8.6 – Comparaison entre les étapes de planification expliquée dans les chapitres précédents (à gauche figure 5.1) et celles de la méthode appliquée dans ce chapitre (à droite figure 8.5)

section 5.2.1.

Exemple 26 $[oui(PU_1) \rightarrow question \rightarrow oui(C(L_3, L_4)) \rightarrow question \rightarrow oui(C(L_1, L_3)) \rightarrow question \rightarrow besoin(police),\ oui(FB_1) \rightarrow question \rightarrow oui(C(L_1, L_0)) \rightarrow question \rightarrow besoin(pompier)] \longrightarrow eteindre\_le\_feu$ *est un plan faisable dans le graphe ET/OU de l'exemple 24. Tous les nœuds appartenant aux plans faisables de ce graphe sont représentés par des nœuds remplis en noir dans la figure 8.4.*

2. Dans chaque plan faisable obtenu, nous éliminons les nœuds de type *"oui(agent)"*, *"question"* et *"besoin(Spec(agent))"*. Nous obtenons des plans appelés **plans faisables exécutables** où tous les nœuds sont exécutables. Dans cette étape, le nombre de nœuds de chaque plan faisable exécutable est au moins divisé par deux par rapport au nombre des nœuds du plan faisable puisqu'on supprime tous les nœuds artificiels.

   Exemple 27 $[oui(C(L_3, L_4)) \rightarrow oui(C(L_1, L_3)),\ oui(C(L_1, L_0))] \longrightarrow eteindre\_le\_feu$ *est le plan faisable exécutable obtenu du plan faisable de l'exemple 26.*

3. Un **plan temporel faisable exécutable** est un plan faisable exécutable où les nœuds sont transformés en actions ou tâches d'exécution. Un nœud $oui(C(L_i, L_j))$ est transformé sous la forme d'une tâche $aller(agent, L_j, L_i)$ avec des données et des contraintes temporelles $\langle I^-_{aller(agent,L_j,L_i)}, I^+_{aller(agent,L_j,L_i)}, \Delta_{aller(agent,L_j,L_i)}, Pr_{aller(agent,L_j,L_i)}, C_{aller(agent,L_j,L_i)} \rangle$.

   Exemple 28 *Le plan faisable exécutable de l'exemple précédent devient :*
   $[aller(PU_3, L_4, L_3) \rightarrow aller(PU_3, L_3, L_1),\ aller(FB_1, L_0, L_1)] \longrightarrow eteindre\_le\_feu$
   où chacune de ces tâches est caractérisée par une liste de données. Par exemple $eteindre\_le\_feu$ *a la liste de données suivante :* $\langle [0, 110], \{60, 90\}, \{0.75, 0.25\}, \{50, 100\} \rangle$. *Les autres listes*

de données sont dans l'annexe A et les plans temporels faisables exécutables de l'exemple précédent sont dans l'annexe B.

4. Afin de sélectionner les **plans admissibles**, le planificateur détermine les intervalles d'exécution de toutes les tâches de chaque plan temporel faisable exécutable, ensuite il vérifie la possibilité d'exécuter les tâches pendant les intervalles d'exécution trouvés. Pour les détails revoir la section 5.2.2.
5. Le **plan-solution** à proposer doit atteindre le but initial, respecter toutes les contraintes et répondre le mieux aux attentes de l'utilisateur. Le planificateur fait la propagation de probabilités sur les intervalles d'exécution possibles et calcule les valeurs espérées de coût et de temps des tâches des plans admissibles pour ensuite sélectionner le meilleur plan-solution selon les critères de l'utilisateur. Rappelons qu'un plan-solution admet plusieurs ordonnancements, c'est pourquoi le planificateur sélectionne **l'ordonnancement-solution** à exécuter. Pour les détails revoir le chapitre 7.

8.2.2.3 Transcription du plan-solution en commandes

Après la sélection du meilleur ordonnancement-solution selon les critères de l'utilisateur, le planificateur le transforme en commandes compréhensibles par les agents, puis les envoie aux agents concernés. Ces commandes comprennent en plus des tâches à exécuter la date de début de l'exécution de ces dernières. Par exemple, ces deux lignes de commandes peuvent être envoyées respectivement aux agents PU_3 et FB_1.

1. $aller(PU_3, L_4, L_3, 1) \rightarrow aller(PU_3, L_3, L_1, 8) \rightarrow eteindre\_le\_feu(L_1, 20)$
2. $aller(FB_1, L_0, L_3, 1) \rightarrow aller(FB_1, L_3, L_1, 9) \rightarrow eteindre\_le\_feu(L_1, 20)$

Chaque agent commence l'exécution de son plan selon la date de début de la première tâche dans le plan. Les dates de début et de fin d'exécution de chaque tâche sont déterminées par le planificateur (l'agent central) en utilisant la méthode de calcul des intervalles d'exécution tout en vérifiant leurs validités. Le planificateur sélectionne ensuite le plan-solution et enfin l'ordonnancement-solution comme décrit dans le chapitre 7 pour le transformer en commandes et les envoyer aux agents concernés par l'exécution.

8.3 Conclusion

Dans ce chapitre, nous avons présentée comment notre approche peut s'appliquer à un exemple de la *Robocup Rescue* qui consiste à développer des agents représentant des équipes de secours intervenant après un tremblement de terre ayant touché une grande ville. Son objectif principal est de maximiser une fonction qui est basée sur un score calculé en fonction du nombre

de civils encore vivants et de leur état de santé. Ensuite, nous avons présenté un domaine possible d'application de notre système de planification. Il consiste en la planification hors ligne pour un ensemble d'agents hétérogènes travaillant ensemble afin de résoudre un ensemble de buts. Étant donnés une carte spécifique d'une ville, des agents repartis sur des lieux de cette ville et un ensemble de buts à atteindre, la question posée est la suivante : quand et comment doit réagir chaque agent afin de satisfaire l'ensemble des buts, tout en respectant toutes les contraintes du domaine ? La méthode utilisée se déroule en trois étapes principales :

1. Transformation de la carte en graphe ET/OU.
2. Planification temporelle et probabiliste.
3. Transformation de l'ordonnancement-solution en commandes compréhensibles par les agents.

La résolution du problème représenté par un graphe ET/OU ($2^{\text{ème}}$ étape) est basée sur la méthode de planification expliquée dans les chapitres précédents avec quelques modifications. En fait, après la génération des plans faisables du graphe ET/OU, le planificateur élimine les nœuds artificiels comme n'ayant pas des données numériques, ensuite il remplace les nœuds de type "$oui(C(L_i, L_j))$" par des tâches exécutables sous la forme $aller(agent, L_j, L_i)$. Enfin il continue les autres étapes (génération des plans admissibles, génération du plan-solution et génération de l'ordonnancement-solution) sans modification de l'algorithme de planification.

Chapitre 9

Tests expérimentaux et outils

Le planificateur que nous proposons dans cette thèse vise à trouver un plan-solution capable de répondre aux attentes de l'utilisateur et de satisfaire toutes les contraintes du domaine de planification. Un tel domaine est représenté par un ensemble de tâches, un ensemble de buts à atteindre et un ensemble de données. L'environnement de planification est caractérisé par une incertitude sur l'exécution des tâches, traduite par une distribution de probabilités sur les durées d'exécution des tâches. Des contraintes temporelles qualitatives de précédence relient les tâches entre elles et d'autres quantitatives restreignent les dates de début et de fin d'exécution des tâches ainsi que leurs durées d'exécution possibles. Nous sommes donc situés dans un environnement temporel et probabiliste. Rappelons que les tâches possèdent aussi des coûts d'exécution dépendant de leurs durées d'exécution. Notre planificateur est (1) correct : l'exécution du plan-solution permet, partant de l'état initial du monde, d'aboutir à un état but, (2) sain : tous les plans qu'il peut produire sont corrects et (3) complet : il produit un plan permettant de résoudre le problème lorsqu'un tel plan existe. Afin de montrer l'efficacité de notre méthode de planification, nous avons fait plusieurs sortes de tests sur nos algorithmes. Plus précisément, les tests ont été faits sur chacune des étapes de planification.

Dans ce chapitre, nous allons présenter et analyser les résultats obtenus. Nous analysons tout d'abord l'utilité de la sélection des plans faisables, puis détaillons les tests reliés aux intervalles d'exécution des tâches et par conséquence aux plans admissibles et aux ordonnancements. Ensuite, nous analysons l'algorithme de l'union d'intervalles et nous terminons par les analyses effectuées sur le planificateur d'agents.

9.1 Plans faisables

La première étape de notre méthode de planification consiste à trouver dans le graphe ET/OU tous les plans faisables, c.-à-d. ceux qui respectent les contraintes de précédence entre les tâches. Rappelons qu'un plan faisable est un sous-graphe du graphe ET/OU allant de l'ensemble des

tâches initiales à un sous-ensemble de tâches finales et dont l'exécution permet de changer l'état actuel pour atteindre un but précis. La méthode de recherche utilisée est celle en chaînage arrière dans le graphe.

L'objectif de la génération des plans faisables est d'éliminer de l'espace de recherche les tâches qui n'appartiennent à aucune solution possible et par suite de diminuer le nombre de tâches à traiter dans les étapes suivantes. Il est toujours préférable d'explorer dans une direction prometteuse afin d'éviter de gaspiller des ressources sur une solution que nous savons être potentiellement mauvaise. La recherche des plans faisables par le parcours du graphe ET/OU en arrière a pour but de ne pas traiter les tâches qui ne peuvent appartenir à aucune solution. Le nombre de plans faisables dépend de plusieurs paramètres :

- le nombre de tâches dans le graphe ET/OU ;
- le profondeur du graphe ET/OU ;
- le nombre de tâches initiales ;
- le nombre de tâches finales ;
- le nombre de contraintes de précédence de type *ET* et *OU* entre les tâches ;

Nous n'avons pas fait de tests pour calculer le nombre de plans faisables obtenus. Ils seraient en effet peu utiles étant donné le grand nombre de paramètres qui jouent un rôle dans la sélection des plans faisables. Par contre, nous avons calculé le temps d'exécution de génération de ces plans qui croît avec l'augmentation des quatre premiers paramètres et diminue avec le dernier.

9.2 Plans admissibles et ordonnancements

La génération des plans admissibles consiste à sélectionner parmi les plans faisables ceux qui respectent les contraintes temporelles sur les tâches. Pour qu'une tâche appartienne à un plan admissible, il faut qu'elle ait au moins une durée qui forme un intervalle d'exécution possible inclus dans la fenêtre temporelle de la tâche.

Le nombre d'intervalles d'exécution de chaque tâche t dépend du nombre de tâches prédécesseurs, du nombre d'intervalles d'exécution de chaque tâche prédécesseur, de la largeur de sa fenêtre temporelle $[I_t^-, I_t^+]$ et de la cardinalité de son ensemble de durées $|\Delta_t|$. Quand une de ces contraintes croît, le nombre d'intervalles d'exécution croît. Le nombre de plans admissibles dans le graphe ET/OU dépend du nombre d'intervalles d'exécution des tâches, du nombre de tâches et du nombre de contraintes de précédence. Ce nombre croît avec le nombre d'intervalles d'exécution des tâches et du nombre de tâches et diminue avec le nombre de contraintes de précédence. Nous avons analysé le nombre d'intervalles d'exécution dans les cas d'un nœud *OU* et d'un nœud *ET* dans la section 5.2.3 du chapitre 5.

Rappelons qu'un ordonnancement d'un plan admissible est la donnée pour chacune de ses tâches d'un de ses intervalles d'exécution possibles durant lequel elle doit s'exécuter. Le nombre

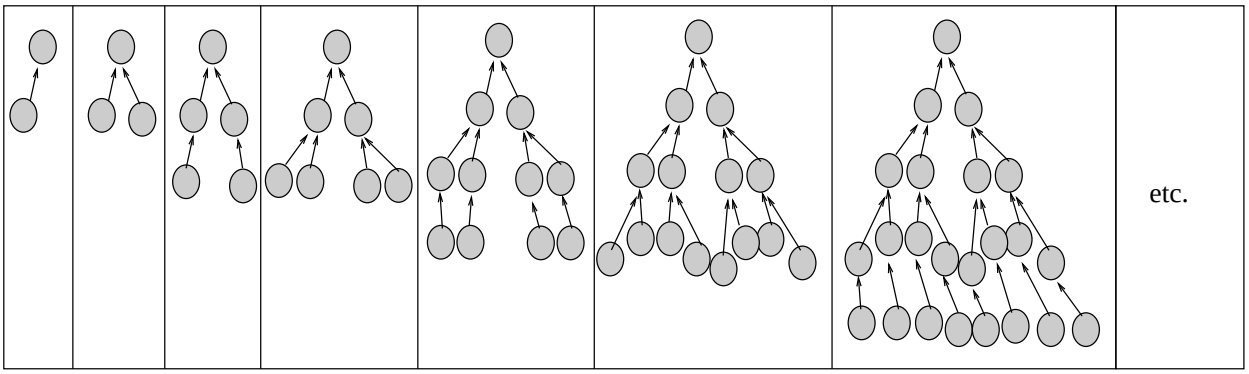

FIG. 9.1 – Des graphes utilisés dans les tests des plans faisables

total des ordonnancements d'un plan $\mathcal{P}_a$ est égal, au pire cas, au produit des cardinalités des ensembles d'intervalles d'exécution possibles de toutes les tâches de $T(\mathcal{P}_a)$.

Nous avons fait deux sortes de tests en ce qui concerne les ordonnancements. Dans le premier, nous avons fixé le nombre de tâches à 6 et nous avons fait varier le nombre d'intervalles d'exécution de ces tâches d'une façon croissante de 1 à 7 intervalles d'exécution par tâche. Dans le deuxième, nous avons fixé le nombre d'intervalles d'exécution pour toutes les tâches (5 intervalles par tâche), puis nous avons augmenté le nombre de tâches dans un graphe ET/OU binaire acyclique. Nous avons commencé la recherche des plans admissibles dans un graphe ET/OU formé de deux tâches seulement (une racine et une feuille) puis nous avons ajouté un successeur à chaque tâche n'ayant qu'un seul successeur. Lorsque toutes les tâches ont 0 ou 2 successeurs, nous ajoutons un successeur à chaque feuille et ainsi de suite. Un exemple de graphes utilisés pour ce test est illustré dans la figure 9.1. Nous avons considéré que toutes les feuilles sont les tâches initiales et que le but est la racine du graphe. Nous constatons que chaque tâche appartient au moins à un plan ce qui signifie que nous allons traiter toutes les tâches avec les contraintes temporelles. Les données temporelles des tâches, à savoir les durées, les dates de début et les dates de fin, sont générées aléatoirement de manière à respecter les fenêtres temporelles.

Le but de ces tests est de calculer le temps d'exécution et le nombre maximal des ordonnancements (combinaison des intervalles d'exécution) si tous les intervalles d'exécution de toutes les tâches sont valides. On calcule également le nombre total d'ordonnancements réellement obtenus après la vérification de la validité des contraintes temporelles. Ceux-ci croissent avec le nombre de tâches, le nombre de buts, la profondeur du graphe ET/OU et le nombre d'intervalles d'exécution des tâches et diminuent avec le nombre des contraintes de précédence entre les tâches.

La figure 9.2 représente les temps d'exécution pour trouver les ordonnancements dans un graphe ET/OU de 6 tâches si nous augmentons respectivement pour chacune des tâches le nombre de ses intervalles d'exécution de 1 à 7. La ligne continue représente le nombre total

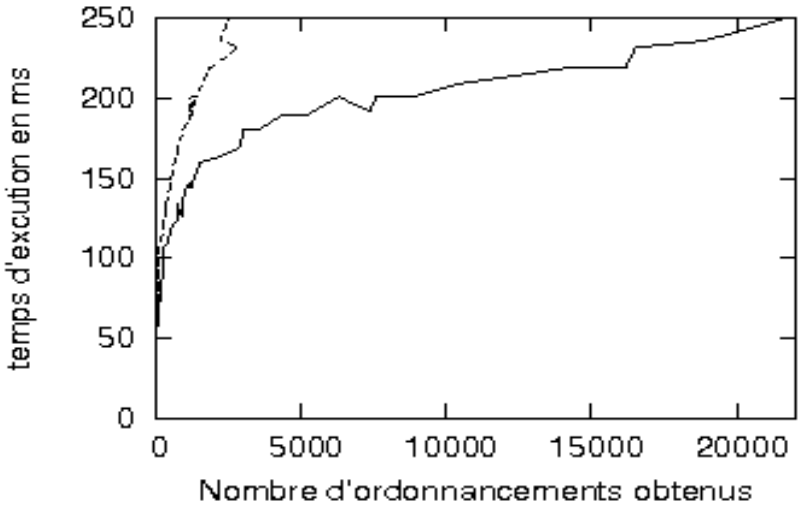

FIG. 9.2 – Les nombres d'ordonnancements obtenus et les nombres d'ordonnancements totaux dans un graphe ET/OU de 6 tâches

d'ordonnancements obtenu si tous les intervalles d'exécution de toutes les tâches sont valides. La ligne pointillée représente le nombre d'ordonnancements obtenu après la suppression de ceux qui ne respectent pas les contraintes temporelles des tâches. Remarquons que dans le premier cas, nous avons obtenu 21609 ordonnancements contre 2506 dans le deuxième pour un temps maximal de 250 ms. Ceci montre l'utilité de cette étape : au lieu de sélectionner le meilleur ordonnancement-solution parmi 21609 ordonnancements possibles, nous allons le sélectionner parmi seulement 2506 ordonnancements.

La partie gauche de la figure 9.3 représente les temps d'exécution pour trouver les ordonnancements des plans admissibles par rapport au nombre des tâches ayant chacune 5 intervalles d'exécution dans le graphe ET/OU. Ce temps est moins d'une seconde pour un graphe ET/OU de 50 tâches. La partie droite de la même figure représente le nombre d'ordonnancements possibles par rapport au nombre des tâches dans le graphe ET/OU. La ligne continue représente le nombre total d'ordonnancements obtenu si tous les intervalles d'exécution de toutes les tâches sont valides. La ligne pointillée représente le nombre d'ordonnancements obtenu après la suppression de ceux qui ne respectent pas les contraintes temporelles des tâches. Nous remarquons que dans le premier cas, le nombre d'ordonnancements croît de façon exponentielle par rapport au nombre des tâches, et que dans le deuxième cas il croît de façon linéaire.

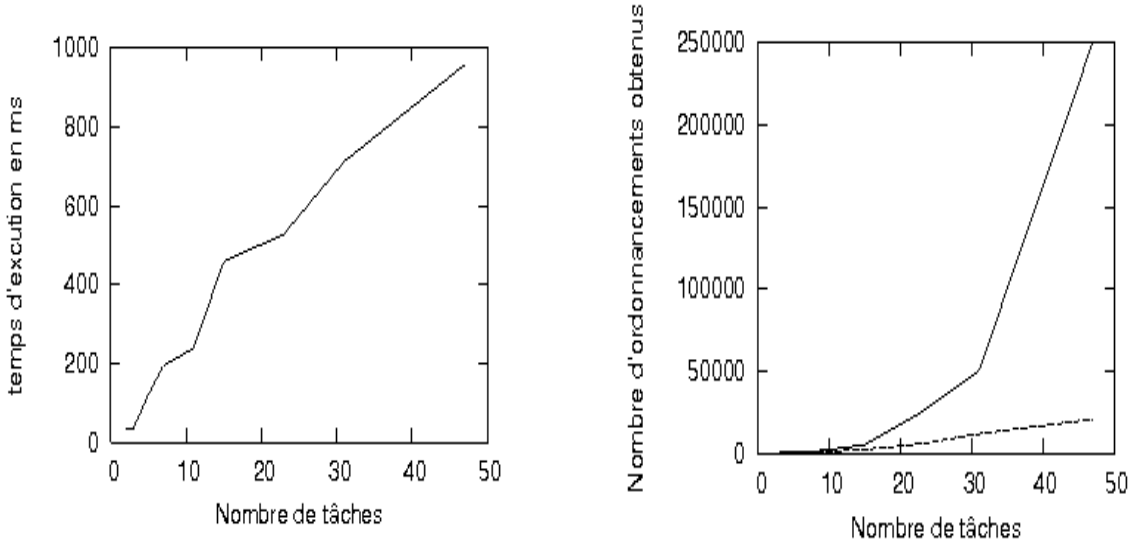

FIG. 9.3 – Les temps d'exécution et les nombres d'ordonnancements obtenus dans un graphe ET/OU (5 intervalles/tâche)

9.3 Unions d'intervalles

L'union des intervalles d'un ordonnancement est l'ensemble des intervalles obtenus en fusionnant dans un même intervalle ceux qui ont un sous-intervalle commun (autrement dit ceux qui se rencontrent).

Nous avons fait deux sortes de tests en ce qui concerne le calcul des unions d'intervalles d'exécution. Nous avons calculé le temps total d'exécution de l'algorithme et le temps d'exécution limité à la fonction qui calcule l'union des intervalles d'exécution, une fois effectué le tri de la liste étiquetée s pour une borne de début et e pour une borne de fin dans l'ordre croissant. Nous n'avons pas trouvé de grande différence entre ces deux temps d'exécution, ce qui signifie que l'étiquetage et le tri de la liste des bornes ne prennent pas beaucoup de temps.

Dans la table 9.1, nous pouvons comparer les différents temps d'exécution en ce qui concerne le temps de calcul étant donnée une liste des bornes triée (colonnes 2 et 6) et le temps total de calcul des unions d'intervalles (colonnes 3 et 7) pour un nombre d'intervalles qui varie de 1 à 1000. Le temps de calcul des unions d'intervalles varie selon qu'ils sont joints, disjoints ou que certains en contiennent d'autres. Mais dans tous les cas, ce temps est raisonnable et ne dépasse pas 1 seconde pour calculer l'union de 1000 intervalles d'exécution. Le temps de l'étiquetage et du tri de la liste des bornes des intervalles est égale à la différence de ces deux temps d'exécution et est représenté dans les colonnes 4 et 8 de la même table.

La figure 9.4 représente les deux courbes des temps d'exécution avec et sans les fonctions

1	2	3	4		5	6	7	8
Nombre d'inter-valles	Temps de calcul des unions en ms	Temps total en ms	temps d'étique-tage et de tri		Nombre d'inter-valles	Temps de calcul des unions en ms	Temps total en ms	temps d'étique-tage et de tri
1	0.064	0.08	0.016		300	50.109	62.097	11.988
10	0.354	0.507	0.153		400	89.275	103.132	13.857
20	0.764	1.085	0.321		500	146.71	162.881	16.171
30	1.279	1.801	0.522		600	203.563	220.548	16.985
40	1.862	2.48	0.618		700	286.143	317.182	31.039
50	2.424	3.296	0.872		800	390.965	415.731	24.766
100	6.657	9.221	2,564		900	567.792	620.578	52.786
200	21.729	26.998	5,269		1000	771.994	830.761	58.767

TAB. 9.1 – Comparaison des temps d'exécution du calcul des unions d'intervalles

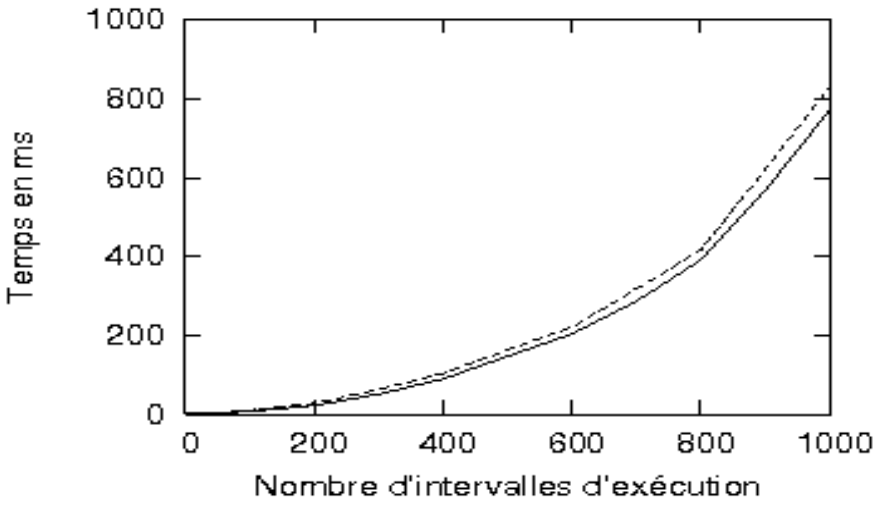

FIG. 9.4 – Les temps de calcul des unions d'intervalles avec et sans la fonction du tri de la liste des bornes

d'étiquetage et du tri de la liste des bornes des intervalles d'exécution. La courbe pointillée représente les temps totaux d'exécution de l'algorithme de calcul des unions d'intervalles pour un nombre d'intervalles variant entre 1 et 1000. La courbe continue représente les temps d'exécution de l'algorithme de calcul des unions d'intervalles après l'étiquetage et le tri de la liste des bornes. La différence entre ces deux courbes représente le temps de l'étiquetage et du tri de la liste des

bornes des intervalles.

9.4 Système de planification de PPA

Le logiciel de planification d'agents que nous avons réalisé se décompose en les parties suivantes :

1. L'interface des données, qui permet de saisir les différentes données et contraintes de l'environnement. Plus précisément, cette interface permet de :
   - créer autant de lieux que l'on désire,
   - créer des chemins entre les lieux et leur ajouter des contraintes,
   - créer des agents sur les lieux et leur attribuer des caractéristiques,
   - créer des buts sur les lieux et leur ajouter des contraintes,
   - charger ou enregistrer une carte,
   - modifier une carte ou n'importe quel objet de la carte.
2. Le système de transformation, qui permet de transformer la carte en un graphe ET/OU de type *ET/OU* et de l'afficher dans une interface permettant de l'enregistrer pour des utilisations ultérieures.
3. Le système de planification, qui analyse le graphe ET/OU obtenu pour en extraire les plans faisables, les plans admissibles, le plan-solution et enfin l'ordonnancement-solution.
4. L'interface résultat, qui permet de visualiser l'ordonnancement-solution trouvé.

La construction de la carte se fait grâce à un éditeur de cartes permettant de créer des lieux et des chemins entre eux. Les agents avec leurs caractéristiques sont créés sur les lieux de la carte. Toutes les données sont entrées dès la création de chaque objet (agent, chemin, but, etc.) et sont stockées dans un fichier de configuration.

La figure 9.5 est une capture d'écran de notre planificateur (PPA). Nous avons illustré sous la forme d'une carte, une partie de la ville de Barcelone en Espagne. Nous trouvons deux postes de police localisés dans les lieux L_7 et L_{12} et deux casernes de pompiers localisées dans les lieux L_5 et L_7. Afin d'atteindre le but localisé dans le lieu L_1, le système de transformation produit tout d'abord un graphe ET/OU comme illustré dans la figure 9.6. Ensuite, le système de planification produit le meilleur plan à exécuter parmi tous les plans admissibles trouvés.

Du point de vue des performances, le système permet de résoudre un problème de planification d'agents représenté dans une carte de très grande taille. La transformation d'une carte en un graphe ET/OU s'effectue de manière quasi-instantanée. Les tests effectués sur les autres étapes de sélection d'un plan-solution sont expliqués dans les sections précédentes.

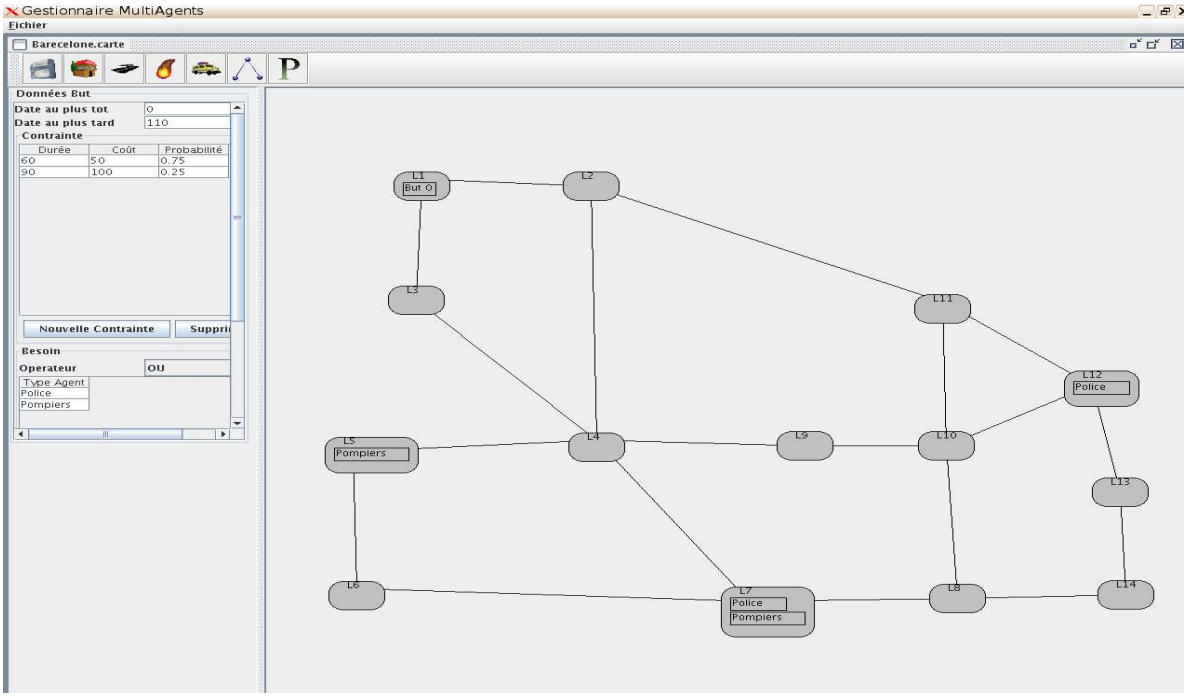

FIG. 9.5 – Capture d'écran d'une carte introduite au planificateur PPA

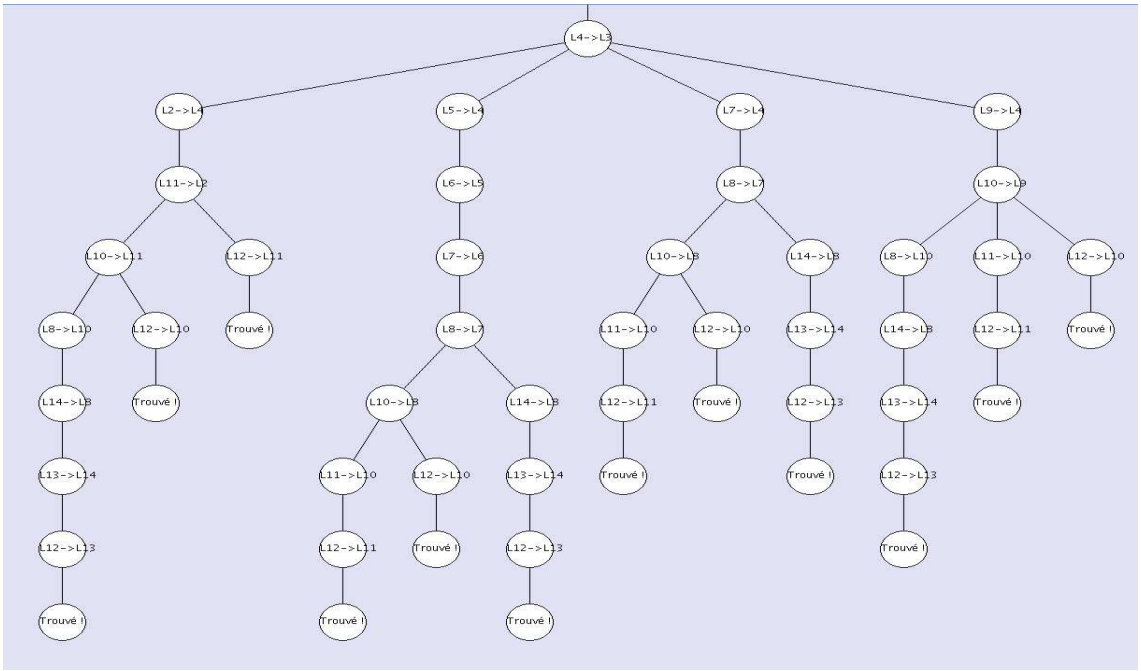

FIG. 9.6 – Capture d'écran d'une partie du graphe ET/OU obtenu à partir de la carte de la figure 9.5

9.5 Conclusion

Dans ce chapitre, nous avons présenté les tests effectués sur notre système de planification. Ces résultats expérimentaux nous confortent dans notre idée du bien-fondé de l'approche. La résolution d'un problème de planification temporel et probabiliste se fait d'une manière quasi-instantanée. Cependant, des tests expérimentaux supplémentaires sur les autres facteurs, comme la taille des fenêtres temporelles et le nombre de contraintes de précédence, restent à réaliser et à analyser.

Troisième partie

Bilan général

Chapitre 10

Conclusion et Perspectives

10.1 Conclusion générale

Nous avons présenté dans ce document une nouvelle approche de la planification temporelle sous incertitude. Les principales contributions concernent les points suivants :

1. représentation de l'environnement dans un graphe ET/OU,
2. les tâches possèdent des contraintes temporelles, des coûts et des probabilités,
3. la durée d'exécution d'une tâche est incertaine,
4. le problème de planification est un problème multi-critères,
5. l'exemple implémenté est un exemple d'application complexe :
   - les agents sont hétérogènes et spécialisés dans des domaines bien précis,
   - pas de communication directe entre les agents,
   - la résolution du but nécessite peut-être plusieurs agents du même type ou de types différents.

Le plan-solution proposé par notre planificateur répond au mieux aux attentes de l'utilisateur. Ainsi, étant donné un problème de planification, notre planificateur détermine le meilleur plan-solution capable d'atteindre les buts en répondant aux critères de préférence de l'utilisateur en terme de temps, coût ou probabilité et en respectant toutes les contraintes de l'environnement. Les contraintes sont temporelles quantitatives (cas des données temporelles numériques comme la date de début, la date de fin et les différentes durées d'exécution possibles des tâches) ou qualitatives (cas des contraintes de précédence entre les tâches, qu'elles soient de type conjonctif (nœud ET) ou de type disjonctif (nœud OU)).

Nous avons présenté une méthode de planification basée sur la propagation des contraintes temporelles et des probabilités dans le graphe ET/OU. Cette technique est composée de plusieurs étapes :

1. La recherche en chaînage arrière des plans faisables. Ces plans sont des sous-graphes du graphe ET/OU ; chacun d'eux est un chemin qui mène de l'état initial à l'état but en respectant la précédence entre les tâches. La technique de la recherche en chaînage arrière (en commençant par les buts) a la particularité d'éviter le parcours des tâches qui ne peuvent appartenir à aucune solution, gagnant ainsi du temps et de la mémoire de CPU.

2. La sélection des plans admissibles à partir de l'ensemble des plans faisables. C'est dans cette étape que nous faisons la propagation des contraintes temporelles en chaînage avant sur les tâches des plans faisables. Nous commençons par les tâches initiales et nous arrêtons quand toutes les tâches sont analysées. Nous calculons niveau par niveau pour chaque tâche de chaque plan faisable les intervalles d'exécution possibles en se basant sur sa fenêtre temporelle, ses durées d'exécution possibles et les intervalles d'exécution possibles des tâches précédentes. Les intervalles d'exécution valides seront analysés dans les étapes suivantes et les autres sont rejetés puisqu'ils ne satisfont pas les contraintes temporelles locales des tâches. Pour qu'un intervalle d'exécution d'une tâche soit valide, il faut que sa date de fin ne dépasse pas la date de fin au plus tard de la fenêtre temporelle de la tâche. Si tous les intervalles d'exécution d'une tâche sont non-valides, la tâche est considérée comme non exécutable et par conséquence le plan faisable qui contient cette tâche est éliminé de l'espace de sélection du plan-solution. L'avantage de cette étape est de diminuer le nombre des plans à analyser dans les étapes suivantes. À chaque fois que nous trouvons une tâche non-valide, nous avons un plan en moins pour faire la propagation de probabilités et de coûts sur ses tâches.

3. La sélection du plan-solution parmi l'ensemble des plans admissibles. C'est ici que les préférences de l'utilisateur entrent en jeu. Tout d'abord, nous faisons la propagation des probabilités sur les intervalles d'exécution possibles, ensuite nous proposons un plan-solution selon les attentes de l'utilisateur. Nous proposons soit le plan le plus probable, soit un plan compromis entre un petit coût et une grande probabilité, soit un plan répondant à un ordre lexicographique sur les préférences de l'utilisateur en ce qui concerne le temps, le coût et la probabilité. Nous sélectionnons ensuite l'ordonnancement-solution selon un des critères cités ci-dessus ou selon l'union des intervalles d'exécution qui a pour but de réduire le nombre des intervalles d'exécution en fusionnant ceux qui sont joints. Le calcul des unions d'intervalles permet de comparer plusieurs ordonnancements afin d'en choisir un seul pour l'exécution. Nous pouvons choisir celui qui commence le plus tôt possible, celui qui a la durée d'exécution minimale, celui qui termine le plus tôt possible, etc.

Nous avons aussi présenté un exemple d'application de notre planificateur. Il s'agit d'une gestion d'une situation de crise où plusieurs agents doivent collaborer pour atteindre un ou plusieurs buts. Le problème, dit problème de planification d'agents, est représenté sous la forme d'une carte

composée de lieux, de chemins, d'un ensemble d'agents hétérogènes spécialisés dans des domaines bien déterminés et distribués dans des groupes de spécialités sur les lieux de la carte et des buts à atteindre dans des certains lieux. La résolution de ce problème se fait via un agent central qui gère, organise et planifie les tâches de l'ensemble des agents. Les contraintes du domaine concernent particulièrement les chemins (distance, blocage, temps, coûts et probabilités de traversée), les agents (disponibilité, spécialité, localisation) et les buts (type, localisation, fenêtre temporelle, durées, coûts et probabilités). Pour résoudre le problème de planification d'agents, l'agent central construit, à partir de la carte, un graphe ET/OU en commençant par les nœuds qui représentent les buts à atteindre jusqu'à ce qu'il trouve les agents concernés par la résolution du problème et qui représentent les feuilles du graphe. Ensuite, il applique la méthode de planification expliquée ci-dessus en générant les plans faisables, les plans admissibles et enfin le plan-solution et son ordonnancement-solution qui sera transformé en commandes compréhensibles par les agents.

Ce type d'application ressemble au domaine de simulation de la *"Robocup Rescue"*. Cependant, étant donné que nous planifions hors ligne et que dans la *"Robocup Rescue"* la planification est en temps réel, la comparaison entre les méthodes de planification n'est pas crédible, d'autant plus que d'autres paramètres propres à chaque système entrent en jeu dans la planification.

Les tests que nous avons effectués ont montré l'efficacité de notre système global ainsi de chaque étape de planification. Le temps de trouver un plan-solution est quasi-instantané, même avec un grand nombre de tâches. Cependant, des tests expérimentaux supplémentaires sur les autres facteurs, comme la profondeur du graphe ET/OU, la taille des fenêtres temporelles et le nombre de contraintes de précédence, sont à réaliser et analyser.

10.2 Perspectives

La méthode de recherche d'un plan-solution que nous utilisons dans cette thèse consiste à trouver tous les plans faisables, propager les contraintes temporelles sur les tâches de ces plans pour obtenir les plans admissibles et enfin maximiser ou minimiser une fonction objectif. L'avantage essentiel de cette méthode est que nous ne propageons les contraintes temporelles (les durées d'exécution et les fenêtres temporelles) que sur les tâches appartenant à une solution possible et nous ne calculons le score de la fonction objectif que pour les plans candidats à être un plan-solution. Son inconvénient est que nous sommes obligés de calculer tous les scores de tous les plans admissibles pour pouvoir parmi ceux-ci choisir le meilleur. Néanmoins, une autre méthode de recherche envisageable consiste à calculer le score d'un seul plan admissible et à le considérer comme un score de comparaison. Ensuite, on calcule le score de chaque plan et si on arrive à un score supérieur au score de comparaison, on arrête l'analyse des tâches contenues dans le plan courant et on passe aux autres plans. Le plan ayant le score minimal est le plan-solution. Dans le pire des cas, nous serons obligés de calculer les scores de tous les plans admissibles. En revanche

il n'y a pas de différence entre cette méthode et la nôtre si nous cherchons à maximiser un score quelconque. Mais il est aussi possible de déterminer le score de choix des plans, par exemple nous pouvons éliminer un plan dès que son score dépasse un certain coût. Étant donné que nous utilisons une méthode de recherche par chaînage arrière et que la propagation des contraintes temporelles se fait par chaînage avant, nous ne pouvons appliquer cette méthode qu'après l'étape de sélection des plans faisables. Une comparaison entre cette méthode et la méthode utilisée dans cette thèse est envisagée pour la suite à court-terme de notre travail.

Nous travaillons dans un environnement incertain où la durée d'exécution de chaque tâche n'est pas connue. Néanmoins chaque tâche ne peut s'exécuter que pendant une des durées qui lui sont attribuées avant le début de la planification, avec des probabilités et des coûts d'exécution. Ces durées sont calculées grâce à des observations et études statistiques antérieures. La méthode de planification que nous proposons dans ce mémoire est fondée sur l'idée de planifier hors ligne, c'est-à-dire que toutes les démarches à faire sont établies avant l'exécution effective des tâches. Or, puisque l'environnement est dynamique, il est possible qu'après l'élaboration du plan-solution, des nouvelles contraintes soient ajoutées au domaine de planification. Dans ce cas le plan-solution proposé n'est plus le meilleur plan. Nous proposons, dans un prolongement futur de ce travail, de vérifier l'exécution du plan-solution et de remédier à son échec en cas de changement de l'environnement. Si l'exécution d'un plan suivant l'ordonnancement choisi devait échouer d'une façon ou d'une autre (par exemple à cause d'une mauvaise prévisibilité des temps d'exécution de certaines de ses tâches), un nouveau plan pour les tâches restantes ainsi qu'un nouvel ordonnancement doit être établi le plus rapidement possible. Nous distinguons quatre sortes d'échec :

1. échec semi-partiel : étant donné que le système de planification propose un ordonnancement-solution composé d'une suite de tâches à exécuter pendant des intervalles d'exécution bien déterminés, il est possible que l'exécution d'une de ses tâches ne respecte pas l'intervalle associé à celle-ci. Si la date de fin de son exécution réelle est inférieure à la date de début de l'intervalle d'exécution de la tâche suivante, l'ordonnancement-solution reste valide, mais n'est plus nécessairement le meilleur ordonnancement-solution. Par contre, si la date de fin de son exécution réelle est supérieure à la date de début de l'intervalle d'exécution de la tâche suivante, nous nous trouvons par conséquent dans un cas où l'exécution de la tâche ne s'est pas déroulée comme prévu. Pour remédier à cela, il faut rétablir un autre ordonnancement avec une nouvelle date de début possible pour la tâche successeur. Nous parlons dans ce cas d'échec semi-partiel.

2. échec partiel : si l'exécution d'une tâche échoue totalement pour une raison ou une autre, nous nous trouvons dans un cas d'échec partiel. Nous proposons d'établir un nouveau plan, en considérant que la ou les tâches prédécesseurs de la tâche qui a échoué sont des tâches

initiales avec des dates de début et de fin égales à l'instant courant et un ensemble de durées d'exécution vide et en éliminant de l'espace de recherche la tâche dont l'exécution a échoué, les tâches déjà exécutées et les tâches reliées directement et exclusivement à celle-ci. S'il est impossible de trouver un plan à partir de cet état, nous nous trouvons dans un cas d'échec semi-total.

3. échec semi-total : si l'exécution d'une tâche échoue dans un plan-solution et s'il est impossible d'établir un autre plan remplaçant le premier, nous considérons que cet état est un échec semi-total où nous ne pouvons rien faire dans l'immédiat : il faut attendre des nouvelles informations ou un changement dans le domaine de planification.
4. échec total : nous sommes dans un échec total si nous ne trouvons pas de plan-solution valide pour résoudre le problème de planification.

Un autre point important qui mériterait une étude ultérieure est d'étudier la possibilité de commencer l'exécution avant que le processus de planification ne soit fini. Ceci a l'avantage de ne pas faire attendre la fin de la planification pour pouvoir commencer l'exécution des tâches. Par conséquent, on évite, si la planification prend beaucoup du temps, un retard de l'exécution des premières tâches conduisant à un échec partiel ou total du plan-solution. Néanmoins dans cette méthode il est difficile de trouver un plan-solution répondant aux critères de préférence de l'utilisateur mais elle permet de bien choisir les tâches l'une après l'autre selon l'état actuel de l'environnement et par conséquent de gagner du temps lors d'un échec. Dans la section 2.10 du chapitre 2, nous avons fait une étude des différents systèmes qui s'appliquent à la planification et à l'exécution. Une idée intéressante est d'utiliser des heuristiques inspirées de ces systèmes afin de réaliser un planificateur exécutif se basant sur notre méthode de planification.

Nous avons appliqué notre planificateur sur un exemple qui est la gestion d'une situation de crise. Les tests effectués ont montré l'efficacité et la rapidité de notre méthode de planification. Or une comparaison avec d'autres systèmes travaillant dans le même domaine comme la *"Robocup Rescue"* est essentielle. Ce qui nous a empêché de faire cette comparaison est la grande différence de la représentation des données, surtout en ce qui concerne la probabilité et le coût, ainsi que le contraste entre la planification en ligne utilisée dans la Robocup Rescue et la planification hors ligne utilisée dans notre système.

Annexes

Annexe A

Les données initiales de l'exemple 24 de la section 8.2 :

$aller(PU, L_3, L_4) < [0, 10], \{5, 7, 10\}, \{0.25, 0.20, 0.55\}, \{5, 10, 10\} >$

$aller(PU, L_0, L_3) < [0, \infty], \{2, 5\}, \{0.55, 0.45\}, \{5, 5\} >$

$aller(PU, L_0, L_1) < [0, \infty], \{5, 9\}, \{0.65, 0.35\}, \{5, 10\} >$

$aller(FB, L_0, L_3) < [0, \infty], \{5, 8\}, \{0.60, 0.40\}, \{5, 10\} >$

$aller(FB, L_1, L_3) < [0, \infty], \{3, 5\}, \{0.20, 0.80\}, \{5, 5\} >$

$aller(PU, L_1, L_3) < [2, \infty], \{7, 9\}, \{0.25, 0.75\}, \{10, 15\} >$

$aller(FB, L_0, L_1) < [0, \infty], \{5, 10\}, \{0.30, 0.70\}, \{10, 20\} >$

$eteindre\_le\_feu < [0, 110], \{60, 90\}, \{0.75, 0.25\}, \{50, 100\} >$

$aller(PU, L_1, L_2) < [1, 15], \{10, 12, 14\}, \{0.10, 0.20, 0.70\}, \{5, 5, 10\} >$

$aller(FB, L_1, L_2) < [1, 15], \{9, 10\}, \{0.45, 0.55\}, \{10, 10\} >$

$aller(FB, L_3, L_4) < [0, 10], \{7, 10\}, \{0.20, 0.80\}, \{5, 15\} >$

Annexe B

Les plans faisables exécutables de l'exemple 24 de la section 8.2 :

$[aller(PU_1, L_0, L_1),\ aller(FB_1, L_0, L_1))] \longrightarrow eteindre\_le\_feu$
$[aller(PU_1, L_0, L_1),\ aller(FB_2, L_0, L_1))] \longrightarrow eteindre\_le\_feu$
$[aller(PU_1, L_0, L_1),\ aller(FB_1, L_0, L_3) \rightarrow aller(FB_1, L_3, L_1)] \longrightarrow eteindre\_le\_feu$
$[aller(PU_1, L_0, L_1),\ aller(FB_2, L_0, L_3) \rightarrow aller(FB_2, L_3, L_1)] \longrightarrow eteindre\_le\_feu$
$[aller(PU_1, L_0, L_3) \rightarrow aller(PU_1, L_3, L_1),\ aller(FB_1, L_0, L_1))] \longrightarrow eteindre\_le\_feu$
$[aller(PU_1, L_0, L_3) \rightarrow aller(PU_1, L_3, L_1),\ aller(FB_2, L_0, L_1))] \longrightarrow eteindre\_le\_feu$
$[aller(PU_1, L_0, L_3) \rightarrow aller(PU_1, L_3, L_1), aller(FB_1, L_0, L_3) \rightarrow aller(FB_1, L_3, L_1)] \longrightarrow eteindre\_le\_feu$
$[aller(PU_1, L_0, L_3) \rightarrow aller(PU_1, L_3, L_1), aller(FB_2, L_0, L_3) \rightarrow aller(FB_2, L_3, L_1)] \longrightarrow eteindre\_le\_feu$

$[aller(PU_2, L_0, L_1),\ aller(FB_1, L_0, L_1))] \longrightarrow eteindre\_le\_feu$
$[aller(PU_2, L_0, L_1),\ aller(FB_2, L_0, L_1))] \longrightarrow eteindre\_le\_feu$
$[aller(PU_2, L_0, L_1),\ aller(FB_1, L_0, L_3) \rightarrow aller(FB_1, L_3, L_1)] \longrightarrow eteindre\_le\_feu$
$[aller(PU_2, L_0, L_1),\ aller(FB_2, L_0, L_3) \rightarrow aller(FB_2, L_3, L_1)] \longrightarrow eteindre\_le\_feu$
$[aller(PU_2, L_0, L_3) \rightarrow aller(PU_2, L_3, L_1),\ aller(FB_1, L_0, L_1))] \longrightarrow eteindre\_le\_feu$
$[aller(PU_2, L_0, L_3) \rightarrow aller(PU_2, L_3, L_1),\ aller(FB_2, L_0, L_1))] \longrightarrow eteindre\_le\_feu$
$[aller(PU_2, L_0, L_3) \rightarrow aller(PU_2, L_3, L_1), aller(FB_1, L_0, L_3) \rightarrow aller(FB_1, L_3, L_1)] \longrightarrow eteindre\_le\_feu$
$[aller(PU_2, L_0, L_3) \rightarrow aller(PU_2, L_3, L_1), aller(FB_2, L_0, L_3) \rightarrow aller(FB_2, L_3, L_1)] \longrightarrow eteindre\_le\_feu$

$[aller(PU_3, L_4, L_3) \rightarrow aller(PU_3, L_3, L_1),\ aller(FB_1, L_0, L_1)] \longrightarrow eteindre\_le\_feu$
$[aller(PU_3, L_4, L_3) \rightarrow aller(PU_3, L_3, L_1),\ aller(FB_2, L_0, L_1)] \longrightarrow eteindre\_le\_feu$
$[aller(PU_3, L_4, L_3) \rightarrow aller(PU_3, L_3, L_1), aller(FB_1, L_0, L_3) \rightarrow aller(FB_1, L_3, L_1)] \longrightarrow eteindre\_le\_feu$
$[aller(PU_3, L_4, L_3) \rightarrow aller(PU_3, L_3, L_1), aller(FB_2, L_0, L_3) \rightarrow aller(FB_2, L_3, L_1)] \longrightarrow eteindre\_le\_feu$
$[aller(PU_3, L_4, L_3) \rightarrow aller(PU_3, L_3, L_0) \rightarrow aller(PU_3, L_0, L_1), aller(FB_1, L_0, L_1)] \longrightarrow eteindre\_le\_feu$
$[aller(PU_3, L_4, L_3) \rightarrow aller(PU_3, L_3, L_0) \rightarrow aller(PU_3, L_0, L_1), aller(FB_2, L_0, L_1)] \longrightarrow eteindre\_le\_feu$
$[aller(PU_3, L_4, L_3) \rightarrow aller(PU_3, L_3, L_0) \rightarrow aller(PU_3, L_0, L_1), aller(FB_1, L_0, L_3) \rightarrow aller(FB_1, L_3, L_1)]$
$\longrightarrow eteindre\_le\_feu$
$[aller(PU_3, L_4, L_3) \rightarrow aller(PU_3, L_3, L_0) \rightarrow aller(PU_3, L_0, L_1), aller(FB_2, L_0, L_3) \rightarrow aller(FB_2, L_3, L_1)]$
$\longrightarrow eteindre\_le\_feu$

Annexe C

Les tables 0.1 et 0.2 représentent la liste des principales notations et symboles que nous utilisons dans notre thèse.

Symbole	Signification
t	tâche
T	ensemble des tâches
T_I	ensemble des tâches initiales
T_M	ensemble des tâches intermédiaires
T_F	ensemble des tâches finales
I_t^-	date de début au plus tôt de la tâche t
I_t^+	date de fin au plus tard de la tâche t
$[I_t^-, I_t^+]$	fenêtre temporelle d'exécution de la tâche t
d_t^i	une durée d'exécution possible de la tâche t
Δ_t	ensemble des durées d'exécution possibles de la tâche t
Pr_t^i	probabilité d'exécution de la tâche t pendant la durée d_t^i
Pr_t	ensemble de probabilités d'exécution de la tâche t
c_t^i	coût d'exécution de la tâche t pendant la durée d_t^i
C_t	ensemble de coûts d'exécution de la tâche t
δ_{t_i,t_j}	délai entre les tâches t_i et t_j
D	ensemble des délais entre les tâches
$c = [t_1, t_2, \ldots, t_m] \rightarrow t$	contrainte de précédence conjonctive
$c = t_1 \vert t_2 \vert \ldots \vert t_m \rightarrow t$	contrainte de précédence disjonctive
$t_i \rightarrow t$	contrainte de précédence simple
$G = (T, E, D)$	graphe de planification

TAB. 0.1 – Liste de notations

Symbole	Signification
I_t^i	un intervalle d'exécution possible de la tâche t
$\mathcal{I}_t$	ensemble d'intervalles d'exécution
s_t^i	une date de début d'exécution possible de t
S_t	ensemble des dates de début d'exécution possibles de t
e_t^i	une date de fin d'exécution possible de t
E_t	ensemble des dates de fin d'exécution possibles de t
$\mathcal{P}_F$	ensemble des plans faisables
$\mathcal{P}_f$	un plan faisable
$\mathcal{P}_A$	ensemble des plans admissibles
$\mathcal{P}_a$	un plan admissible
$pr_{debut}(s_t^r \mid e_{t'}^j)$	la probabilité que l'exécution de t commence à s_t^r sachant que son prédécesseur t' a terminé son exécution à $e_{t'}^j$
$Pr(I_t^r \mid e_{t'}^j)$	la probabilité de l'exécution de l'intervalle I_t^r de la tâche t sachant que son prédécesseur t' a terminé son exécution à $e_{t'}^j$
$\vartheta(cout(t))$	la valeur espérée de coût de t
$\vartheta(temps(t))$	la valeur espérée de temps de t
$Pr(\mathcal{P}_a)$	la probabilité d'exécution d'un plan admissible $\mathcal{P}_a$
$\mathcal{P}^s$	le plan-solution
$\vartheta(cout(\mathcal{P}_a))$	la valeur espérée totale de coût du plan $\mathcal{P}_a$
$\vartheta(temps(\mathcal{P}_a))$	la valeur espérée totale de temps du plan $\mathcal{P}_a$
$\mathcal{P}_{ord}$	un ordonnancement possible
$ORD(\mathcal{P}_a)$	l'ensemble des ordonnancements possibles du plan $\mathcal{P}_a$
$\mathcal{P}_{ord}^s$	l'ordonnancement-solution
$\vartheta(cout(\mathcal{P}_{ord}))$	la valeur espérée totale de coût de l'ordonnancement $\mathcal{P}_{ord}$
$\vartheta(temps(\mathcal{P}_{ord}))$	la valeur espérée totale de temps de l'ordonnancement $\mathcal{P}_{ord}$
$Union(Int(\mathcal{P}_{ord}))$	l'ensemble d'unions d'intervalles
$U_i = [s_i, e_i]$	une union d'intervalles

TAB. 0.2 – Suite de la liste de notations

Bibliographie

[Aberdeen *et al.* 04] D. Aberdeen, S. Thiébaux & L. Zhang, « Decision-Theoretic Military Operations Planning. », *in Proceedings of the Fourteenth International Conference on Automated Planning and Scheduling - ICAPS 2004*, pages 402–412, 2004.

[Aikat 96] S. Aikat, « Using PERT to Plan and Schedule Your Documentation Projects », *Society for Technical Communication*, 1996.

[Albers 97] P. Albers, *IxTeT : extension de la représentation pour la prise en compte des effets dépendant du contexte et des axiomes du domaine.*, Thèse de doctorat, Laboratoire d'Analyse et d'Architecture des Systèmes, Toulouse, Juillet 1997.

[Allen & Hamilton 64] Allen & Hamilton, *New Uses and Management Implications of PERT Booz*, Allen and Hamilton, New York, 1964.

[Allen 83] J. F. Allen, « Maintaining Knowledge about Temporal Intervals. », *Commun. ACM*, vol. 26, n° 11, 832–843, 1983.

[Allen 81] J. F. Allen, « An Interval-Based Representation of Temporal Knowledge. », *in IJCAI*, pages 221–226, Canada, August Proceedings of the 7th International Joint Conference on Artificial Intelligence, IJCAI '81.

[Baki 05] B. Baki, « Ordonnancement et planification sous contraintes temporelles et probabilistes. », *in 3ème manifestation des jeunes chercheurs en Sciences et Technologies de l'Information et de la Communication, (MAJECSTIC 2005)*, pages 323–330, Rennes, France, 2005.

[Baki & Beynier 04] B. Baki & A. Beynier, « Planification Probabiliste et Temporelle. », *in 2ème manifestation des jeunes chercheurs en Sciences et Technologies de l'Information et de la Communication, (MAJECSTIC 2004)*, Calais, France, 2004.

[Baki *et al.* 04] B. Baki, A. Beynier, M. Bouzid & A. I. Mouaddib, « Temporal Probabilistic Task Planning in an Emergency Service », *in Complex Systems Intelligence and Modern Technological Applications, (CSIMTA 2004)*, pages 427–434, Cherbourg, France, 2004.

[Baki & Bouzid 05a] B. Baki & M. Bouzid, « Probabilistic Planning with Temporal Constraints », *in International Congress on Computational Intelligence,(ICCI 2005)*, pages 8–15, Monteria, Colombia, 2005.

[Baki & Bouzid 05b] B. Baki & M. Bouzid, « A scheduling technique of plans with probability and temporal constraints. », *in Proceedings of the Second International Conference on Informatics in Control, Automation and Robotics, (ICINCO 2005)*, pages 77–84, Barcelona, Spain, 2005.

[Baki & Bouzid 05c] B. Baki & M. Bouzid, « Scheduling with Probability and Temporal Constraints. », *in 9th Congress of the Italian Association for Artificial Intelligence, (AI*IA 2005)*, vol. 3673, pages 148–159, Milan, Italy, 2005.

[Baki & Bouzid 06] B. Baki & M. Bouzid, « Planification et ordonnancement probabilistes sous contraintes temporelles. », *in 15ème congrès francophone Reconnaissance des Formes et Intelligence Artificielle (RFIA 2006)*, page 99, Tours, France, 2006.

[Baki *et al.* 05] B. Baki, M. Bouzid & A. Ligeza, « Generating Low Cost Plans under Uncertainty and Temporal Constraints. », *in 17th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2005)*, pages 531–536, Hong Kong, China, 2005.

[Baki *et al.* 06] B. Baki, M. Bouzid, A. Ligeza & A.-I. Mouaddib, « A centralized planning technique with temporal constraints and uncertainty for multi-agent systems », *Journal of Experimental and Theoretical Artificial Intelligence (JETAI)*, 2006.

[BASTIÉ 97] C. BASTIÉ, *Intégration de la planification et du suivi d'exécution d'actions parallèles :le système SPEEDY*, Thèse de doctorat, Université paul Sabatier, Toulouse, Septembre 1997.

[Bellman 57] R. Bellman, *Dynamic Programming*, Princeton University Press, 1957.

[Besse-Patin 05] C. Besse-Patin, « Planification, ordonnancement et apprentissage par renforcement », Examen de synthèse, Université Laval, Canada, 2005.

[Bessière 92] C. Bessière, *Systèmes à contraintes évolutifs en Intelligence Artificielle.*, Thèse de doctorat, Université de Montpellier II, septembre 1992.

[Beynier 03] A. Beynier, « MDPs interactifs et contraintes temporelles », Rapport technique, Université de Caen, 2003.

[Beynier & Mouaddib 05] A. Beynier & A.-I. Mouaddib, « A polynomial algorithm for Decentralized Markov Decision Processes with temporal constraints. », *in Proceedings of the 4th International joint Conference on Autonomous Agents and Multi Agent Systems (AAMAS-05)*, pages 963–969, Utrecht, The Netherlands, 2005.

[Beynier & Mouaddib 06] A. Beynier & A.-I. Mouaddib, « An iterative algorithm for solving Constrained Decentralized Markov Decision Processes. », *in Proceedings of the Twenty-First National Conference on Artificial Intelligence (AAAI-06)*, Boston, Massachusetts, 2006.

[Blum & Furst 97] A. Blum & M. Furst, « Fast planning through planning graph analysis », *Artif. Intell.*, vol. 90, 281–300, 1997.

[Blum & Langford 99] A. Blum & J. Langford, « Probabilistic Planning in the Graphplan Framework. », *in 5th European Conference on Planning, ECP'99*, pages 319–332, September 1999.

[Bonet & Geffner 03] B. Bonet & H. Geffner, « Labeled RTDP : Improving the Convergence of Real-Time Dynamic Programming. », *in Proceedings of the Thirteenth International Conference on Automated Planning and Scheduling - ICAPS 2003-*, pages 12–31, 2003.

[Bouzid 95] M. Bouzid, *Contribution au Raisonnement Temporel : Application aux Systèmes de Maintien de Vérité.*, Thèse de doctorat, Université Henri Poincaré - Nancy I, juillet 1995.

[Bresina & Washington 00] J. Bresina & R. Washington, « Expected Utility Distributions for Flexible Contingent Execution », *in Proceedings of the AAAI-2000 Workshop : Representation Issues for Real-World Planning Systems*, 2000.

[Bérubé 03] J.-F. Bérubé, « Planification pour agents dans un environnement dynamique et incertain », Août 2003.

[Cayrol *et al.* 00] M. Cayrol, P. Régnier & V. Vidal, « LCGP : Une amélioration de Graphplan par relâchement de contraintes entre actions simultanées. », *in Actes du 12ème Congrès de Reconnaissance des Formes et Intelligence Artificielle, RFIA-2000*, pages 79–88, Paris, France, 2000.

[Chapman 87] D. Chapman, « Planning for Conjunctive Goals. », *Artif. Intell.*, vol. 32, n° 3, 333–377, 1987.

[Dechter *et al.* 91] R. Dechter, I. Meiri & J. Pearl, « Temporal Constraint Network », *Artificial Intelligence*, vol. 49, n° 1-3, 61–95, 1991.

[Do & Kambhampati 03] M. B. Do & S. Kambhampati, « Sapa : A Multi-objective Metric Temporal Planner. », *J. Artif. Intell. Res. (JAIR)*, vol. 20, 155–194, 2003.

[Draper *et al.* 94] D. Draper, S. Hanks & D. Weld, « Probabilistic Planning with Information Gathering and Contingent Execution », *in Proceedings of the Second International Conference on AI Planning Systems, AIPS-94*, pages 31–36, Chicago, June 1994.

[Dvorak *et al.* 04] Z. Dvorak, D. Král & O. Pangrác, « Locally Consistent Constraint Satisfaction Problems : (Extended Abstract) », *in Proceedings of the 31st International Colloquium Automata, Languages and Programming, ICALP*, vol. 3142, pages 469–480, 2004.

[Erol *et al.* 94] K. Erol, J. Hendler & D. S. Nau, « HTN Planning : Complexity and Expressivity », *in Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*, vol. 2, pages 1123–1128, Seattle, Washington, USA, 1994, AAAI Press/MIT Press.

[Esquirol & Lopez 99] P. Esquirol & Lopez, « L'ordonnancement », Rapport technique, LAAS, Economica, Paris, 1999.

[Farreny 95] H. Farreny, *Recherche heuristiquement ordonnée dans les graphes d'états : Algorithmes et propriétés*, Editions MASSON, Paris, 1995.

[Fikes & Nilsson 71] R. Fikes & N. J. Nilsson, « STRIPS : A New Approach to the Application of Theorem Proving to Problem Solving », *Artif. Intell.*, vol. 2, n° 3/4, 189–208, 1971.

[Fourman 00] M. P. Fourman, « Propositional Planning », *in Workshop on Model-Theoretic Aproaches to Planning, AIPS 2000*, April 2000. {Available as EDI-INF-RR-0034}

[Fox & Long 98] M. Fox & D. Long, « The Automatic Inference of State Invariants in TIM. », *J. Artif. Intell. Res. (JAIR)*, vol. 9, 367–421, 1998.

[Fox & Long 99] M. Fox & D. Long, « The Detection and Exploitation of Symmetry in Planning Problems. », *in Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence, IJCAI 99*, pages 956–961, 1999.

[Gaborit 96] P. Gaborit, *Planification Distribuée pour la coopération multi-agents*, Thèse de doctorat, Université paul Sabatier, Toulouse, Septembre 1996.

[Gantt] « http ://www.ilog.fr/products/ganttnet/frenchdemos.cfm », .

[Garcia *et al.* 00] F. Garcia, J. Lang & A.-I. Mouaddib, *Processus Décisionnel de Markov*, Temps et évolution, Cépaduès-Edition, 2000.

[Ghallab *et al.* 04] M. Ghallab, D. Nau & P. Traverso, *Automated Planning, Theory and Practice*, Elsevier, Morgan Kaufmann Publishers, May 2004.

[Guere & Alami 99] E. Guere & R. Alami, « A Possibilistic Planner that Deals with Non-Determinism and Contingency. », *in Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence, IJCAI 99*, pages 996–1001, 1999.

[Hanna 03] H. Hanna, *La planification des tâches et des ressources sous incertitude dans un système multi-agent.*, Thèse de doctorat, Université de Caen, décembre 2003.

[Hayashi *et al.* 04] H. Hayashi, K. Cho & A. Ohsuga, « A New HTN Planning Framework for Agents in Dynamic Environments. », *in 4th International Workshop of Computational Logic in Multi-Agent Systems, CLIMA IV*, vol. 3259 de *Lecture Notes in Computer Science*, pages 108–133, 2004.

[Howard 60] R. A. Howard, *Dynamic Programming and Markov Processes*, MIT Press, Cambridge, Massachussets, 1960.

[Joslin & Pollack 96] D. Joslin & M. E. Pollack, « Is "Early Commitment" in Plan Generation Ever a Good Idea ? », *in AAAI/IAAI, Vol. 2*, pages 1188–1193, 1996.

[Kambhampati 97] S. Kambhampati, « Refinement Planning as a Unifying Framework for Plan Synthesis. », *AI Magazine*, vol. 18, n° 2, 67–97, 1997.

[Kambhampati 99] S. Kambhampati, « Improving Graphplan's Search with EBL & DDB Techniques. », *in Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence, IJCAI 99*, pages 982–987, 1999.

[Kautz & Selman 99] H. A. Kautz & B. Selman, « Unifying SAT-based and Graph-based Planning. », *in Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence, IJCAI 99*, pages 318–325, 1999.

[Koehler 98] J. Koehler, « Planning under Resource Constraints. », *in ECAI*, pages 489–493, 1998.

[Koehler *et al.* 97] J. Koehler, B. Nebel, J. Hoffmann & Y. Dimopoulos, « Extending Planning Graphs to an ADL Subset. », *in Proceedings of the 4th European Conference on Planning, ECP*, vol. 1348, pages 273–285, 1997.

[Korf 87] R. E. Korf, « Planning as Search : A Quantitative Approach. », *Artif. Intell.*, vol. 33, n° 1, 65–88, 1987.

[Kushmerick *et al.* 94] N. Kushmerick, S. Hanks & D. S. Weld, « An Algorithm for Probabilistic Least-Commitment Planning. », *in AAAI*, pages 1073–1078, 1994.

[Kushmerick *et al.* 95] N. Kushmerick, S. Hanks & D. S. Weld, « An Algorithm for Probabilistic Planning. », *Artif. Intell.*, vol. 76, n° 1-2, 239–286, 1995.

[Laborie 95] P. Laborie, *IxTeT : Une approche intégrée pour la gestion de ressources et la synthèse de plans*, Thèse de doctorat, Ecole Nationale Supérieure des Télécommunications, Paris, 1995.

[Laruelle 94] H. Laruelle, *Planification temporelle et exécution de tâches en robotique*, Thèse de doctorat, Université Paul Sabatier, Toulouse, Avril 1994.

[Le-Gloannec *et al.* 05] S. Le-Gloannec, A.-I. Mouaddib & F. Charpillet, « A Decision-Theoretic Scheduling of Resource-Bounded Agents in Dynamic Environments. », *in Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, Monterey California, US, 2005.

[Lemai & Ingrand 04] S. Lemai & F. Ingrand, « Interleaving Temporal Planning and Execution in Robotics Domains. », *in Proceedings of the Nineteenth National Conference on Artificial Intelligence, Sixteenth Conference on Innovative Applications of Artificial Intelligence, AAAI*, pages 617–622, July 2004.

[Lever & Richards 94] J. M. Lever & B. Richards, « parcPlan : A Planning Architecture with Parallel Actions, Resources and Constraints. », *in Methodologies for Intelligent Systems, 8th International Symposium, ISMIS '94*, vol. 869 de *Lecture Notes in Computer Science*, pages 213–222, October 1994.

[Little 04] I. Little, « Probabilistic temporal planning », Master, The Australian National University, Department of Computer Science, 2004.

[Little *et al.* 05] I. Little, D. Aberdeen & S. Thiébaux, « Prottle : A Probabilistic Temporal Planner. », *in Proceedings of the Twentieth National Conference on Artificial Intelligence and the Seventeenth Innovative Applications of Artificial Intelligence Conference,AAAI*, pages 1181–1186, July 2005.

[Littman 99] M. L. Littman, « Initial Experiments in Stochastic Satisfiability. », *in AAAI/IAAI*, pages 667–672, 1999.

[Long & Fox 99] D. Long & M. Fox, « Efficient Implementation of the Plan Graph in STAN. », *J. Artif. Intell. Res. (JAIR)*, vol. 10, 87–115, 1999.

[Lopez & Roubellat 01] P. Lopez & F. Roubellat, *Ordonnancement de la production.*, Hermès Science publications, Paris, 2001.

[Majercik & Littman 98] S. M. Majercik & M. L. Littman, « MAXPLAN : A New Approach to Probabilistic Planning. », *in AIPS*, pages 86–93, 1998.

[McAllester & Rosenblitt 91] D. A. McAllester & D. Rosenblitt, « Systematic Nonlinear Planning. », *in AAAI*, pages 634–639, 1991.

[McDermott 82] D. V. McDermott, « A Temporal Logic for Reasoning About Processes and Plans. », *Cognitive Science*, vol. 6, 101–155, 1982.

[Melnik *et al.* 02] S. Melnik, H. Garcia-Molina & E. Rahm, « Similarity Flooding : A Versatile Graph Matching Algorithm and Its Application to Schema Matching. », *in Proceedings of the 18th International Conference on Data Engineering, ICDE*, pages 117–128, San Jose, CA, 2002.

[Mouaddib & Zilberstein 97] A.-I. Mouaddib & S. Zilberstein, « Handling Duration Uncertainty in Meta-Level Control of Progressive Processing. », *in IJCAI*, pages 1201–1207, 1997.

[Nebel *et al.* 97] B. Nebel, Y. Dimopoulos & J. Koehler, « Ignoring Irrelevant Facts and Operators in Plan Generation. », *in Proceedings of the 4th European Conference on Planning, ECP*, vol. 1348, pages 338–350, 1997.

[Nguyen & Kambhampati 00] X. Nguyen & S. Kambhampati, « Extracting Effective and Admissible State Space Heuristics from the Planning Graph. », *in Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on on Innovative Applications of Artificial Intelligence,AAAI/IAAI*, pages 798–805, July 2000.

[Nguyen & Kambhampati 01] X. Nguyen & S. Kambhampati, « Reviving Partial Order Planning. », *in Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence, IJCAI 2001*, pages 459–466, 2001.

[Nigenda *et al.* 00] R. S. Nigenda, X. Nguyen & S. Kambhampati, « AltAlt : Combining the advantages of Graphplan and Heuristic State Search. », Rapport technique, ASU, 2000.

[Onder & Pollack 97] N. Onder & M. E. Pollack, « Contingency Selection in Plan Generation. », *in 4th European Conference on Planning, ECP'97*, vol. 1348 de *Lecture Notes in Computer Science*, pages 364–376, September 1997.

[Onder & Pollack 99] N. Onder & M. E. Pollack, « Conditional, Probabilistic Planning : A Unifying Algorithm and Effective Search Control Mechanisms. », *in proceedings of the 15th National Conference on AI, AAAI/IAAI*, pages 577–584, Madison, 1999.

[Onder *et al.* 04] N. Onder, G. C. Whelan & L. Li, « Probapop : Probabilistic Partial-Order Planning. », *in The Probabilistic Planning Track of the International Planning Competition, IPC-4*, Canada, June 2004.

[Onder *et al.* 06] N. Onder, G. C. Whelan & L. Li, « Engineering a Conformant Probabilistic Planner. », *J. Artif. Intell. Res. (JAIR)*, vol. 25, 1–15, 2006.

[P. Fournier-Viger and L. Lebel] « www.dmi.usherb.ca/~fournier/plplan_fr.html », .

[Pednault 89] E. Pednault, « ADL : Exploring the middle ground between STRIPS and the situation calculus », *in Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning*, pages 324–332, 1989.

[Penberthy & Weld 92] J. S. Penberthy & D. S. Weld, « UCPOP : A Sound, Complete, Partial Order Planner for ADL. », *in KR*, pages 103–114, 1992.

[Penberthy & Weld 94] J. S. Penberthy & D. S. Weld, « Temporal Planning with Continuous Change. », *in AAAI*, pages 1010–1015, 1994.

[Peot & Smith 92] M. Peot & D. Smith, « Conditional nonlinear planning », *in* J. Hendler (dir.), *Proceedings of the First International Conference on AI Planning Systems*, pages 189–197, College Park, Maryland, June 15–17 1992, Morgan Kaufmann.

[Pollack & Horty 99] M. Pollack & J. F. Horty, « There's More to Life than Making Plans », *The AI Magazine*, vol. 20, n° 4, 71–84, 1999.

[Puterman 94] M. L. Puterman, *Markov Decision Processes : Discrete Stochastic Dynamic Programming*, John Wiley and Sons, New York, 1994.

[Robocup Rescue] « http ://www.rescuesystem.org/robocuprescue/ », .

[Roy & Dibon 66] B. Roy & M. Dibon, « L'ordonnancement par la méthode des potentiels - Le programme CONCORD », *Automatisme*, pages 1–11, février 1966.

[Russell & Norvig 03] S. J. Russell & P. Norvig, *Artificial Intelligence : A Modern Approach.*, Prentice Hall, 2nde édition, 2003.

[Régnier 90] P. Régnier, « Planification : historique, principes, problèmes et méthodes (de GPS à ABTWEAK) », Rapport technique, IRIT, Université Paul Sabatier, Toulouse, 1990.

[Salles 04] M. Salles, « Planification spatio-temporelle et situations de crise », Rapport technique, Université de Caen, 2004.

[Sapena & Onaindia 02] O. Sapena & E. Onaindia, « Domain-Independent Online Planning for STRIPS Domains. », *in 8th Ibero-American Conference on AI, IBERAMIA*, pages 825–834, Spain, 2002.

[Smith *et al.* 00] D. Smith, J. Frank & A. Jonsson, « Bridging the Gap Between Planning and Scheduling », 2000.

[Smith & Weld 98] D. E. Smith & D. S. Weld, « Conformant Graphplan. », *in AAAI/IAAI*, pages 889–896, 1998.

[Smith & Weld 99] D. E. Smith & D. S. Weld, « Temporal Planning with Mutual Exclusion Reasoning. », *in Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence, IJCAI 99*, pages 326–337, 1999.

[Vere 85] S. Vere, « Temporal Scope of Assertions and Window Cutoff. », *in IJCAI*, pages 1055–1059, 1985.

[Vere 81] S. A. Vere, « Planning in time : windows and durations for activities and goals. », Rapport technique, Jet Propulsion Laboratory, Pasadena, 1981.

[Vidal 95] T. Vidal, *Le Temps en Planification et Ordonnancement : vers une gestion complète et efficace de contraintes hétérogènes et entachées d'incertitude.*, Thèse de doctorat, Université paul Sabatier, Toulouse, Septembre 1995.

[Vidal *et al.* 99] V. Vidal, M. Cayrol & P. Régnier, « Contribution à l'amélioration de GRAPHPLAN Formalisation, critique, modification : LCGP (Volume 1) », Rapport technique, IRIT, Université Paul Sabatier, 1999.

[Vidal & Geffner 04a] V. Vidal & H. Geffner, « Branching and Pruning : An Optimal Temporal POCL Planner Based on Constraint Programming. », *in Proceedings of the Nineteenth National Conference on Artificial Intelligence, Sixteenth Conference on Innovative Applications of Artificial Intelligence, AAAI*, pages 570–577, USA, July 2004.

[Vidal & Geffner 04b] V. Vidal & H. Geffner, « Un planificateur temporel optimal basé sur la programmation par contraintes. », *in Actes des Dixièmes Journées Nationales sur la résolution de problèmes NP-Complets (JNPC-2004)*, pages 347–362, Angers, France, 2004.

[Warren 76] D. H. D. Warren, « Generating Conditional Plans and Programs. », *in proccedings of the AISB Summer conference (ECAI)*, pages 344–354, Edinburg, 1976.

[Weiss 99] G. Weiss, *Multiagent Systems : A Modern Approach to Distributed Artificial Intelligence*, MIT Press, 1999.

[Weld 94] D. S. Weld, « An Introduction to Least Commitment Planning », *AI Magazine*, vol. 15, n° 4, 27–61, 1994.

[Weld *et al.* 98] D. S. Weld, C. R. Anderson & D. E. Smith, « Extending Graphplan to Handle Uncertainty and Sensing Actions », *in AAAI/IAAI*, pages 897–904, 1998.

[Wilkins 88] D. E. Wilkins, *Practical planning : extending the classical AI planning paradigm*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1988.

[Zimmerman & Kambhampati 99] T. Zimmerman & S. Kambhampati, « Exploiting Symmetry in the Planning graph via Explanation-Guided Search. », *in AAAI/IAAI*, pages 605–611, 1999.