Intégration des ressources en planification temporelle optimale

Thesis · December 2008

DOI: 10.13140/RG.2.1.3790.6967

1 author:

Zied Loukil
University of Sfax
19 PUBLICATIONS **23** CITATIONS


Université d'Artois
Centre de recherche CRIL-CNRS

INTEGRATION DES RESSOURCES EN PLANIFICATION TEMPORELLE OPTIMALE

THÈSE

Présentée et soutenue publiquement le 12 décembre 2008
à la faculté de sciences Jean Perrin de Lens
en vue de l'obtention du

Doctorat de l'Université d'Artois

(Spécialité informatique)

par

Zied LOUKIL

Composition du jury :

Rapporteurs :	Monsieur Abdel-Illah MOUADDIB
	Monsieur Bertrand NEVEU
Examinateurs :	Monsieur Lakhdar SAIS
	Monsieur Abdelmajid BEN HAMADOU
	Monsieur Philippe LABORIE
Directeur de thèse :	Monsieur Pierre MARQUIS
Co-directeur de thèse :	Monsieur Vincent VIDAL

DEDICACES

Je dédie ce travail à mon épouse Aida, mon fils Karim et à mes parents qui ont fait tous les sacrifices imaginables pour m'aider et m'encourager dans cette thèse. Qu'il soit à la hauteur de leur estimation.

REMERCIEMENTS

Mes plus vifs remerciements sont adressés à Monsieur Abdelmajid BEN HAMADOU pour son aide inestimable ainsi qu'à Messieurs Pierre MARQUIS et Vincent VIDAL sans qui je n'aurais jamais réussi à réaliser ce travail. Je n'oublierai jamais ce qu'ils ont fait pour moi.

J'adresse tous mes remerciements aussi à Monsieur Abdel-Illah MOUADDIB, Monsieur Philippe LABORIE et Monsieur Bertrand NEVEU pour l'intérêt qu'ils ont porté à mon travail en acceptant d'être membres dans le comité d'évaluation.

Je remercie aussi mes sœurs, mes belles-sœurs, ma belle-mère et mon beau-frère pour leur encouragement et leur soutien.

Je tiens enfin à remercier tous mes amis et amies dans les laboratoires CRIL et MIRACL ainsi qu'à l'Université d'Artois et l'Université de Sfax pour leur réconfort, leur soutien et l'ambiance géniale qu'ils entretiennent.

TABLE DES MATIERES

INTRODUCTION

La planification classique indépendante du domaine peut être classée parmi les principaux thèmes de recherche en intelligence artificielle. Elle vise à trouver une liste ordonnée des actions à effectuer par un système donné pour qu'il arrive à passer de son état initial à un état but initialement précisé.

Le problème d'existence d'un plan valide en STRIPS propositionnel est *PSPACE*-complet [Bylander 1991]; ce qui rend le problème de la planification classique indépendante du domaine difficile à résoudre dans le cas général. Cependant, grâce aux travaux de recherche successifs, plusieurs planificateurs permettant de résoudre des instances de plus en plus complexes dans ce domaine ont été développés. Ils ont permis de traiter des problèmes proches de la réalité en passant de la planification classique au sens strict - qui est la classe la plus restrictive dans laquelle les problèmes de planification sont très loin du monde réel – à des extensions de celle-ci plus expressives et donc plus adaptés à la résolution des problèmes réels.

Parmi les formalismes d'extension de la planification classique, on peut citer la planification temporelle, une classe particulière de problèmes de planification indépendante du domaine dans laquelle sont considérées des actions à durées différentes. Elle permet de prendre en considération le temps et même les ressources, ce qui rend ses problèmes proches de ceux de l'ordonnancement. Plusieurs planificateurs temporels ont été développés, dont certains permettent de gérer le temps et les ressources comme par exemple le planificateur TP4[Geffner et Haslum 2001] et d'autres gèrent le temps sans tenir compte des ressources comme le cas du planificateur CPT[Geffner et Vidal 2004 ; Geffner et Vidal 2005]. La recherche dans ce domaine est encore intense vu la richesse et la diversité des méthodes à utiliser et l'insuffisance des résultats obtenus.

C'est dans ce dernier contexte que s'inscrivent les travaux que nous présentons dans ce mémoire de thèse. Il s'agit précisément d'étudier les relations entre le temps et les ressources dans la planification temporelle indépendante du domaine en vue de proposer une approche d'intégration des ressources consommables et renouvelables dans la planification temporelle.

Le formalisme que nous avons établi pour la description des ressources est inspiré des travaux réalisés dans la gestion des ressources en ordonnancement et en planification. Il est compatible avec le formalisme de Smith est Weld [Smith et Weld 1999] de représentation du temps basé sur l'exclusion mutuelle. Nous avons ensuite élaboré deux approches pour la gestion des ressources : la première permet de trouver des plans valides (s'il en existe) dans un minimum de temps, mais elle impose des restrictions assez sévères. La deuxième approche, moins restrictive que la première, consomme parfois un temps de recherche plus long pour trouver des plans valides d'une meilleure qualité.

Pour expérimenter notre travail, nous avons choisi comme d'étendre le planificateur CPT, qui est un planificateur temporel optimal parallèle indépendant du domaine et qui se fond sur le formalisme de Smith et Weld pour exprimer le temps. Dans sa version originale, ce planificateur ne permet pas de gérer les ressources dans les problèmes qu'il traite, ce qui constitue son inconvénient majeur malgré ses grandes performances qui l'ont distingué parmi les autres planificateurs dans son domaine dans les compétitions de planification IPC4, IPC5 et IPC6. En effet, les plans trouvés par la version originale de ce planificateur peuvent être invalides si on prend les ressources en considération.

Notre principal apport est donc de donner au planificateur CPT la capacité de gérer les ressources par consommation et par production tout en gardant la puissance et les performances de ce planificateur à gérer le temps. Le critère d'optimisation utilisé est la durée totale du plan construit.

Le présent mémoire est organisé en cinq chapitres.

Le premier chapitre constitue une introduction aux concepts de la planification classique. Nous y présentons aussi deux formalismes très utilisés pour la description des problèmes de planification classique indépendante du domaine à savoir le formalisme STRIPS et le

formalisme PDDL. Enfin, nous y présentons les méthodes de recherche les plus utilisées dans cette classe de problèmes de planification en détaillant, pour chacune, certains planificateurs.

Une des extensions les plus importantes de la planification classique est la planification temporelle, obtenue par l'introduction de données temporelles et éventuellement de données numériques relatives aux ressources. Cette extension est introduite dans le deuxième chapitre en deux parties centrées respectivement sur temps et sur les ressources.

Dans la première partie, nous introduisons certains formalismes de représentation du temps dans la planification temporelle notamment le formalisme PDDL2.1, qui est l'extension du formalisme PDDL pour représenter les données temporelles. Nous présentons aussi dans cette partie quelques planificateurs temporels parmi les plus remarquables développés dans ce domaine, notamment le planificateur CPT.

Dans la deuxième partie, nous effectuons une étude comparative de la représentation des ressources en ordonnancement et en planification utilisée dans les planificateurs temporels présentés dans la première partie.

Le troisième chapitre montre comment il est possible de pallier l'un des inconvénients majeurs du planificateur temporel CPT (i.e. son incapacité de gérer les ressources). Nous présentons ainsi un formalisme que nous avons élaboré pour la représentation des ressources dans ce planificateur. Il s'inspire de l'efficacité de la gestion des ressources dans les problèmes d'ordonnancement et de la puissance expressive de la représentation des ressources dans les problèmes de planification temporelle. En plus, le formalisme élaboré est compatible avec le formalisme de Smith et Weld basé sur l'exclusion mutuelle pour la représentation des données temporelles et utilisé par le planificateur CPT. Nous présentons à la fin du chapitre les contraintes de ressources exigées par le formalisme élaboré.

Dans le quatrième chapitre, nous présentons une première méthode pour la gestion des ressources respectant le formalisme décrit dans le troisième chapitre. Cette méthode, basée sur un nouveau concept appelé liens de ressources, exige certaines restrictions et des contraintes supplémentaires limitant les possibilités de parallélisme pour les actions consommant ou produisant la même ressource.

Une étude expérimentale pour la méthode basée sur les liens de ressources est réalisée à la fin de ce chapitre en utilisant le problème ZenoTravel dans sa version temporelle. Cette

étude vérifie, entre autres, la préservation de toutes les performances initiales du planificateur CPT pour traiter les problèmes de planification temporelle sans ressources.

Dans le dernier chapitre, nous présentons une seconde méthode pour la gestion des ressources dans le planificateur CPT. Cette méthode, basée sur le nouveau concept d'actions de synchronisation, est moins restrictive que celle basée sur les liens de ressources. Elle offre donc de nouvelles possibilités pour le parallélisme entre les actions consommant ou produisant la même ressource ; ce qui permet, dans certains cas, d'améliorer la qualité des plans trouvés.

Cette méthode introduit un certain nombre d'actions fictives qui ont l'inconvénient d'augmenter la complexité du problème de planification ; et, par conséquent, du temps de recherche de plans valides et/ou optimaux. Pour limiter cette augmentation du nombre d'actions, nous élaborons certaines règles de propagation, relatives aux actions de synchronisation, qui doivent être introduites dans le planificateur CPT pour réduire l'espace de recherche avant même de commencer le branchement. Ces règles permettent d'éliminer les branches, pour lesquelles, on constate l'incapacité d'avoir des solutions valides.

Ce mémoire renferme une introduction générale et une conclusion générale clôturée par des perspectives de recherche sur la gestion des ressources en planification temporelle, qui se révèle être un domaine très riche dans lequel la recherche reste prometteuse.

CHAPITRE 1

INTRODUCTION A LA PLANIFICATION CLASSIQUE

INTRODUCTION

La planification peut être considérée comme l'un des domaines les plus importants en intelligence artificielle vu l'intensité de la recherche qui ne cesse d'augmenter dans ce domaine depuis près d'un demi-siècle. L'importance de ces recherches se manifeste dans les planificateurs qui ont gagné en performance et en capacité de traiter des problèmes de plus en plus complexes.

Dans ce chapitre, nous allons introduire la planification classique par la présentation des principaux travaux de recherche réalisés concernant la représentation des problèmes de planification et les algorithmes servant à calculer un plan valide (et parfois même optimal selon certains critères).

1. ELEMENTS DE LA PLANIFICATION CLASSIQUE

1.1. Caractéristiques d'un problème de planification classique

Les problèmes considérés comme appartenant à la classe de planification appelée planification classique ont en commun les caractéristiques suivantes :

- le système est markovien et stationnaire et il n'a pas de dynamique propre.
- l'état initial du système est parfaitement connu.
- les actions disponibles sont déterministes et leur effet est parfaitement connu.
- l'observabilité est nulle.
- la satisfaction du but est binaire.
- pour les planificateurs classiques linéaires, un plan solution (ou plan valide) est une suite finie d'actions conduisant à un état but, c'est-à-dire un état dans lequel le but est totalement satisfait. On dit que le plan solution est linéaire. Dans ce cas, le critère déterminant la qualité d'un plan valide (critère d'optimisation) est le nombre d'actions à exécuter dans ce plan.
- pour les planificateurs classiques parallèles, un plan valide est une suite d'étapes successives permettant d'atteindre le but. Chaque étape représente un ensemble d'actions et le passage d'une étape à une autre se fait par l'exécution des actions de l'étape courante. Dans ce

cas, le critère déterminant la qualité d'un plan valide est le nombre d'étapes composant ce plan.

1.2. Définitions formelles

Définition 1.1

Un état du système $\mathcal{S}_i$ est la description de ce système à l'instant i. Dans les formalismes basés sur la logique propositionnelle, il peut être représenté par un n-uplet de variables propositionnelles (appelées fluents).

Dans la planification, il existe deux états particuliers qui sont l'état initial noté $\mathcal{S}_0$, représentant la description initiale du système et l'état but noté $\mathcal{G}$, qui représente une description du système satisfaisant le but et est l'état vers lequel on cherche à faire évoluer le système.

Remarque 1.1

Dans certains contextes de planification, on peut supposer que $\mathcal{G}$ est unique sans perte de généralité.

L'ensemble de tous les états possibles dans un problème de planification est noté $\mathcal{S}$.

Définition 1.2

Les actions constituent les moyens mis à la disposition de l'agent pour faire évoluer le système d'un état à un autre, elles permettent de changer les valeurs des fluents selon leurs effets.

Chaque action peut nécessiter des valeurs particulières pour certains fluents afin de pouvoir être exécutée. Ces valeurs sont appelées des préconditions.

Une action doit donc avoir un ensemble de préconditions et un ensemble d'effets.

La suite d'actions permettant de faire évoluer le système de l'état initial à l'état but est appelée plan linéaire.

Un problème de planification $\mathcal{P}$ peut être défini par le triplet ($\mathcal{S}_0$, $\mathcal{G}$, $\mathcal{A}$) où $\mathcal{S}_0$ représente l'état initial du système, $\mathcal{G}$ représente le but visé et $\mathcal{A}$ représente l'ensemble de toutes les actions applicables au système donné.

1.3. Exemple illustratif

Le problème ZenoTravel[1] est utilisé comme l'un des benchmarks servant à évaluer les performances des planificateurs dans les compétitions IPC. Il s'agit d'utiliser un certain nombre d'avions pour transporter des voyageurs des villes dans lesquelles ils sont situés vers celles auxquelles ils veulent aller.

On suppose avoir dans notre instance de problème un avion A_1, deux personnes P_1 et P_2, deux villes V_1 et V_2 et cinq niveaux de disponibilité du carburant dans le réservoir F_0, F_1, F_2, F_3 et F_4.

Un état du système est décrit par la position de chaque voyageur du problème (dans quelle ville ou quel avion se trouve-t-il ?), la position de chaque avion (dans quelle ville se trouve-t-il ?) et le niveau de carburant dans son réservoir.

Dans notre cas, on suppose avoir dans l'état initial : P_1 dans V_1, P_2 dans V_1, A_1 dans V_2 avec un niveau de carburant F_0. En plus, il faut préciser la succession des niveaux de carburant du problème : F_1 suit F_0, F_2 suit F_1, F_3 suit F_2 et F_4 suit F_3 avec $F_4 < F_3 < F_2 < F_1 < F_0$.

Les actions possibles dans ce problème sont :

- Les actions de la forme *Board_*A_i*_*P_j*_*V_k qui permettent d'embarquer la personne P_j située dans la ville V_k dans l'avion A_i. Pour que l'exécution de l'une de ces actions soit possible, l'avion A_i et la personne P_j doivent être dans la ville V_k.

- Les actions de la forme *Debark_*A_i*_*P_j*_*V_k qui permettent de débarquer la personne P_j située dans l'avion A_i en dehors de celui-ci et dans la ville V_k. Pour que l'exécution de l'une

[1] http://planning.cis.strath.ac.uk/competition/domains.html

de ces actions soit possible, l'avion A_i doit être dans la ville V_k et la personne P_j doit se trouver dans l'avion A_i.

- Les actions de la forme *Fly_A_i_V_j_V_k_F_m_F_n* qui permettent de faire voler l'avion A_i d'une ville V_j dans laquelle il est situé avec une niveau de carburant dans le réservoir F_m, vers une ville V_k à une vitesse normale avec une consommation normale. L'avion devient alors situé à la ville V_k et son niveau de carburant dans le réservoir passe de F_m à F_n. Pour que cette action soit possible, l'avion doit être situé à la ville V_j et le niveau de carburant dans son réservoir doit être F_m. En plus, il faut, dans la succession des niveaux de carburant, avoir F_n qui suit F_m et $F_n < F_m$.

- Les actions de la forme *Zoom_A_i_V_j_V_k_F_l_F_m_F_n* qui permettent de faire voler l'avion A_i d'une ville V_j dans laquelle il est situé avec une niveau de carburant dans le réservoir F_m, vers une ville V_k à une vitesse supérieure avec une consommation double. L'avion devient alors situé à la ville V_k et son niveau de carburant dans le réservoir passe de F_l à F_m ensuite à F_n. Pour que cette action soit possible, l'avion doit être situé à la ville V_j et le niveau de carburant dans son réservoir doit être F_l. En plus, il faut, dans la succession des niveaux de carburant, avoir F_n qui suit F_m, F_m suit F_l et $F_n < F_m < F_l$.

- Les actions de la forme *Refuel_A_i_V_j_F_m_F_n* qui permettent, pour un avion A_i situé dans la ville V_j, de remplir son réservoir de carburant en faisant passer son niveau de F_m à F_n. Pour que cette action soit possible, l'avion A_i doit être situé dans la ville V_j et le niveau de carburant dans son réservoir doit être F_m. En plus, il faut, dans la succession des niveaux de carburant, avoir F_m qui suit F_n et $F_m < F_n$.

Le but du problème est de transporter chaque voyageur vers la ville dans laquelle il veut aller. On peut aussi exiger que l'avion soit posé à la fin dans une ville particulière. Pour cela, le planificateur doit chercher une suite (finie) d'actions faisant évoluer le système de l'état initial à l'état but tout en respectant les préconditions de chaque action.

2. DESCRIPTION DES PROBLEMES DE PLANIFICATION

Les notions décrites précédemment et qui constituent les entrées et les sorties d'un problème de planification doivent être représentées pour être traitées par la machine. En particulier :

- les états (notamment l'état initial et l'état but)
- les fluents
- les actions

Différents choix de représentation peuvent être effectués, on a alors besoin d'un compromis entre l'expressivité des langages retenus (en particulier, la nature des structures préférentielles, la concision des représentations) et l'efficacité des traitements réalisables sur ceux-ci.

Parmi les langages de représentation on peut citer les langages STRIPS [Fikes et Nikson 1971], ADL [Pednault 1989] et PDDL [McDermott 1998].

2.1. Le langage STRIPS

Le formalisme STRIPS [Fikes et Nikson 1971], basé sur la logique propositionnelle est l'une des représentations les plus utilisées pour les problèmes de planification, dans laquelle un état du système est représenté par un ensemble de variables propositionnelles (en considérant que la sémantique de toute variable non représentée est *FAUX*), les états buts sont ceux qui satisfont une conjonction donnée de variables propositionnelles et on représente une action α par un triplet d'ensembles des fluents qui sont:

- **les préconditions** : comme précédemment indiqué, l'exécutabilité d'une action α peut nécessiter des valeurs particulières pour certains fluents ; les uns doivent avoir la valeur *VRAI* et les autres doivent avoir la valeur *FAUX*. L'ensemble des fluents affectés à *VRAI* représente les préconditions de α, il est noté $\mathcal{P}rec(\alpha)$.

- **la liste des ajouts** : après exécution de l'action, certains fluents auront la valeur *VRAI*, l'ensemble de ces fluents, noté $add(\alpha)$, représente la liste des ajouts de α

- **la liste des retraits** : après exécution de l'action, certains fluents auront la valeur *FAUX*, l'ensemble de ces fluents, noté $del(\alpha)$, représente la liste des retraits de α.

Un problème de planification classique $\mathcal{P}$ est donc défini dans le formalisme STRIPS par un triplet (S_0, G, A) où S_0 et G sont deux états particuliers dont le premier représente l'état initial et le deuxième est le but. Chacun d'eux est constitué d'un ensemble de variables propositionnelles représentant les fluents vérifiés (dont la valeur est *VRAI*). A représente l'ensemble d'actions permettant de faire évoluer le système d'un état à un autre.

Une action $\alpha \in A$ est représentée par un triplet $(Prec(\alpha), add(\alpha), del(\alpha))$ où $Prec(\alpha)$, $add(\alpha)$ et $del(\alpha)$ sont trois ensembles de fluents représentant respectivement les préconditions, la liste des ajouts et la liste des retraits relatifs à α.

Un état S_i est représenté par un ensemble de variables propositionnelles indiquant les fluents vérifiés à cet état.

Une action α est applicable à un état S_i si $Prec(\alpha) \subseteq S_i$ et le résultat de l'exécution de α dans ce cas est $Res(S_i,\alpha) = (S_i \setminus del(\alpha)) \cup add(\alpha)$.

Exemple : codage du problème ZenoTravel en STRIPS

L'annexe A contient un codage d'une instance du problème ZenoTravel précédemment cité dans le formalisme STRIPS avec un avion, deux personnes et cinq niveaux de disponibilité du réservoir pour carburant.

Cette manière de représenter les problèmes de planification est facile à interpréter par un planificateur, mais elle est trop lourde à coder pour la représentation des données numériques car on doit attribuer une variable propositionnelle à chaque valeur, ce qui peut donner un grand nombre d'actions à représenter. Par exemple, dans notre problème (qui est relativement simple), nous avons seulement 5 valeurs pour représenter tous les niveaux possibles de carburant dans le réservoir de l'avion, nous avons ainsi attribué un prédicat relatif à chaque valeur pour indiquer qu'elle représente un niveau de carburant et nous avons donné des prédicats supplémentaires pour indiquer la succession entre ces valeurs. Pour avoir des

possibilités plus étendues pour être plus fidèle au monde réel (capacité maximale du réservoir, niveau minimal de carburant pour effectuer un vol, temps de vol en fonction de la distance et de la vitesse du vol,…), nous devons avoir un prédicat pour chaque valeur donnée pour indiquer « l'objet » qu'elle représente et nous devons en plus, avoir un grand nombre de prédicats supplémentaires pour indiquer de simples relations et opérations mathématiques comme les relations d'ordre ou les opérations d'addition ou de soustraction, ce qui risque d'engendrer rapidement une explosion du codage.

Pour cette raison, on peut constater que le langage STRIPS n'est pas adapté aux problèmes avancés de planification comme ceux de la planification temporelle ou de la planification avec ressources.

2.2. Le langage PDDL

Le langage PDDL (Planning Domain Description Language) [McDermott 1998] a été défini en 1998 à l'occasion de la première compétition des planificateurs IPC1 dans le but de développer un seul langage de description des domaines de planification utilisé par tous les planificateurs de cette compétition. Il est ensuite devenu un « standard » utilisé par la plupart des planificateurs. Il peut être considéré comme une forme d'enrichissement du formalisme STRIPS exploitant le pouvoir d'expressivité de la logique du premier ordre ainsi que les propriétés du domaine tout en restant à un niveau symbolique, ce qui permet de représenter de façon concise et claire des problèmes de planification. L'enrichissement se manifeste particulièrement dans :

- la décomposition de l'ensemble des préconditions en deux sous-ensembles, l'un pour les fluents qui doivent avoir la valeur *VRAI*, l'autre pour les fluents qui doivent avoir la valeur *FAUX*.

- l'expression des effets conditionnels, c'est-à-dire des effets qui dépendent de l'état dans lequel l'action est exécutée.

- l'expression des préconditions disjonctives où l'action peut être appliquée à un état si au moins un des fluents de la disjonction est présent dans cet état.

- la possibilité d'utilisation des quantificateurs existentiels (∃) et universels (∀) dans la représentation des préconditions et des effets des actions.

L'enrichissement du langage STRIPS de cette façon-là (en restant à un niveau symbolique) permet de représenter de façon plus concise et plus claire des problèmes de planification.

Exemple : codage du problème ZenoTravel en PDDL

L'annexe B contient un codage du problème ZenoTravel dans le formalisme PDDL :

- Le premier codage (voir Annexe B.1) représente les données communes aux différentes instances du problème, à savoir les composants du système ainsi que la description paramétrée des actions à considérer.

- Le deuxième codage (voir Annexe B.2) représente une instance du problème ZenoTravel avec la description des variables à considérer, l'état initial et l'état but. La complexité de chaque problème est étroitement liée au contenu de l'instance traitée. L'instance représentée dans l'annexe B.2 contient un seul avion, deux villes, deux passagers et cinq niveaux de carburant successifs.

Dans plusieurs problèmes de planification, on remarque que le codage en langage PDDL offre un gain considérable en taille de code par rapport à celle en langage STRIPS. Cependant, la représentation des données numériques comme le temps et les ressources reste encore difficile et assez limitée, comme dans l'exemple ZenoTravel où il faut un prédicat pour chaque niveau de carburant et il faut en plus préciser la succession des différents niveaux.

De plus, il n'est pas possible d'utiliser des opérateurs de comparaison qui seraient utiles pour contrôler le niveau de carburant. Nous pouvons ainsi constater l'importance de l'introduction des données numériques dans la planification pour élargir son horizon.

3. RECHERCHE DE PLANS VALIDES

A partir d'un état initial, d'un ensemble d'actions et d'un but, un planificateur doit fournir une suite d'actions permettant par leur exécution de faire évoluer l'état initial du système vers un état qui satisfait le but.

Plusieurs méthodes de recherche ont été développées comme la recherche par progression, la recherche par régression (qui correspondent respectivement aux notions classiques de chaînage avant et de chaînage arrière), la planification de style Graphplan, la recherche dans l'espace des plans partiels, la planification par satisfaction des formules propositionnelles ou encore la planification par satisfaction de contraintes. Ces méthodes permettent non seulement de trouver des plans séquentiels valides mais aussi des plans parallèles où certaines actions peuvent s'exécuter simultanément si elles ne sont pas interdépendantes.

3.1. Recherche par progression

La recherche de plans valides peut s'effectuer par progression via une recherche dans l'ensemble des états quel que soit le formalisme retenu, pour peu que l'on sache calculer en temps fini pour tout état $\mathcal{S}_i \in \mathcal{S}$ et pour toute action α l'évolution du système de l'état $\mathcal{S}_i$ vers un autre état $\mathcal{P}rog(\mathcal{S}_i, \alpha)$. Cet état est obtenu par l'exécution de l'action α lorsque le système est dans l'état $\mathcal{S}_i$. Il est aussi nécessaire de pouvoir tester en un temps fini si l'état obtenu coïncide avec l'état but.

La planification par progression peut être améliorée en utilisant des fonctions heuristiques dans les algorithmes de recherche, comme il a été observé pour certains des planificateurs les plus performants, en particulier HSP [Bonet et Geffner 1999] et FF [Hoffman et Nebel 2001].

3.1.1. Le planificateur HSP (Heuristic Search Planner)

La recherche heuristique est basée sur le principe de considérer en priorité les actions qui intuitivement conduisent à un état qu'on peut qualifier de « proche » d'un état but (c'est-à-dire pour lequel il restera un nombre minimum d'opérateurs à appliquer), ce qui permet dans un grand nombre de cas de minimiser l'espace de recherche parcouru. En effet, la recherche est

d'autant plus rapide que la qualité de la fonction heuristique utilisée pour calculer la distance entre chaque état et l'état but est pertinente.

Plusieurs planificateurs sont basés sur la recherche heuristique parmi lesquels on peut citer HSP [Bonet et Geffner 1999] qui est un planificateur qui applique une heuristique pour la recherche par progression basée sur l'estimation de la difficulté d'obtenir un plan en effectuant une relaxation du problème de planification : il s'agit de transformer le problème initial de planification $\mathcal{P}$ en un problème relaxé $\mathcal{P}^{\star}$ dans lequel les listes des retraits des actions pour tous les opérateurs sont ignorées. L'heuristique h($\mathcal{S}_i$) calcule ensuite pour chaque état $\mathcal{S}_i \in \mathcal{S}$ une estimation de la distance (nombre d'étapes) qui le sépare du but avec l'estimation de la difficulté de valider chaque atome propositionnel qui le constitue.

HSP utilise ensuite une variante de l'algorithme de recherche locale Hill-Climbing pour trouver des plans valides en choisissant les actions à exécuter à chaque état d'une manière guidée par la fonction heuristique. Pour les actions candidates ayant la même valeur donnée par la fonction heuristique, le choix se fait d'une façon aléatoire.

Le planificateur HSP a obtenu des résultats intéressants notamment dans la première compétition de planification IPC1 [McDermott 2000]. Depuis, il a subi plusieurs modifications et plusieurs variantes sont apparues comme HSP* [Geffner et Haslum 2000] et HSP2.0 [Bonet et Geffner 2001].

3.1.2. Le planificateur FF (Fast-Forward)

Le planificateur FF [Hoffman et Nebel 2001] a une grande ressemblance avec le planificateur HSP. Il traite les problèmes de planification codés en STRIPS ou PDDL et il est basé sur un algorithme de recherche par progression utilisant plusieurs fonctions heuristiques.

L'heuristique principale est celle de HSP basée sur la relaxation du problème de planification en ignorant les listes des retraits des actions pour tous les opérateurs. Alors que HSP utilise une technique d'estimation pour déterminer les distances entre les états et le but dans le problème relaxé (noté $\mathcal{P}^{+}$), FF essaie d'extraire une solution pour $\mathcal{P}^{+}$ en utilisant un algorithme du style Graphplan et le nombre d'actions dans les solutions relaxées trouvées est utilisé comme une estimation de la distance jusqu'au but.

Cette estimation de la distance utilise une stratégie de recherche locale basée sur l'algorithme Enforced Hill-Climbing, une variante de l'algorithme Hill-Climbing qui utilise la

méthode Breadth-First Search pour trouver le meilleur successeur en cas d'états ayant la même évaluation heuristique.

Les plans relaxés sont ensuite utilisés pour élaguer l'espace de recherche grâce à l'heuristique Helpful Actions Prunning qui vise à réduire les successeurs de chaque état à ceux produits par les éléments des plans relaxés.

FF profite du fait que grâce à ses fonctions heuristiques, il est rarement confronté aux problèmes constatés dans la recherche locale (par exemple les minimums locaux) dans les benchmarks des compétitions de planification. De ce fait il a obtenu plusieurs résultats intéressants pour les catégories dans lesquelles il participait ; il a été ainsi récompensé pour ses performances remarquables lors des compétitions internationales de planification IPC2 en 2000 et IPC3 en 2002.

3.2. Recherche par régression

La recherche de plans valides peut s'effectuer par régression via une recherche dans l'ensemble des parties $2^{\mathcal{S}}$ des états quel que soit le formalisme retenu pour peu que l'on sache calculer en temps fini pour tout état $\mathcal{S}_i$ et pour toute action α, l'ensemble des états $\mathcal{E}=\mathcal{R}eg(\mathcal{S}_i,\alpha)$ tel que : $\forall\ \mathcal{S}_j \in \mathcal{E}$ on a $\mathcal{P}rog(\mathcal{S}_j, \alpha)=\mathcal{S}_i$ et tester en temps fini si $\mathcal{E}$ contient l'état initial pour tout $\mathcal{E}\in 2^{\mathcal{S}}$.

Plusieurs planificateurs sont basés sur la recherche par régression parmi lesquels on peut citer le planificateur HSP2.0 [Bonet et Geffner 2001].

Le planificateur HSP2.0

Lorsqu'un plan de coût minimal est recherché (et que les actions ont des coûts positifs qui s'agrègent additivement), on peut utiliser l'algorithme A*: à chaque état $\mathcal{S}_i\in\mathcal{S}$, on associe un coût $f(\mathcal{S}_i) = g(\mathcal{S}_i) + h(\mathcal{S}_i)$ avec $f(\mathcal{S}_i)$ est le coût estimé du nœud $\mathcal{S}_i$ (c'est-à-dire le coût d'un plan de coût minimal passant par l'état $\mathcal{S}_i$), $g(\mathcal{S}_i)$ est le coût réel d'un plan de coût minimal permettant d'atteindre l'état $\mathcal{S}_i$ en partant de l'état de départ $\mathcal{S}_0$ et $h(\mathcal{S}_i)$ est le coût estimé d'un plan de coût minimal permettant d'atteindre un état but en partant de l'état $\mathcal{S}_i$.

HSP2.0 [Bonet et Geffner 2001] est un planificateur qui utilise l'algorithme de recherche WA*, une variante de l'algorithme A* dans laquelle la fonction $h(\mathcal{S}_i)$ (de la fonction

$f(\mathcal{S}_i)=g(\mathcal{S}_i)+h(\mathcal{S}_i)$) est multipliée par le paramètre W≥1. Ce paramètre donne un compromis entre l'optimalité et la vitesse de l'algorithme à trouver une solution:

- Pour W=1, WA* est équivalent à A* et il est optimal tant que la fonction heuristique est admissible.

- Pour W>1, WA* permet souvent de trouver une solution plus rapidement mais la solution trouvée est au plus W fois loin de l'optimalité.

HSP2.0 lit les problèmes de planification codés en PDDL. Il effectue une recherche par progression ou par régression et il peut utiliser les heuristiques h_{add} et h_{max} qui donnent une estimation sur la fonction de coût optimal pour une relaxation du problème de planification dans laquelle les listes de retraits sont ignorées. Ces heuristiques sont obtenues en combinant l'estimation des coûts $g(p, \mathcal{S}_i)$ nécessaires pour atteindre chaque atome p du but à partir de l'état $\mathcal{S}_i$.

3.3. Planification de style Graphplan

Tous les planificateurs du style Graphplan sont des variantes opérant de la même façon que le planificateur originel qui est un algorithme complet (s'il s'arrête sans trouver de solution c'est que le problème n'en admet aucune) et optimal (ici on parle de l'optimalité en ce qui concerne le nombre de niveaux d'actions parallèles) développé en 1995 [Blum et Furst 1995]. Il fonctionne en deux phases : l'expansion du graphe de planification et la recherche du plan solution.

3.3.1. Phase d'expansion du graphe de planification

Le graphe de planification est un graphe orienté représenté par une double structure contenant la liste des actions et celle des fluents. Il est développé durant la phase d'expansion par un mécanisme de chaînage avant commençant par l'état initial du système (noté niveau 0).

L'algorithme procède comme suit :

- pour un niveau k, l'expansion du niveau suivant (k+1) nécessite l'introduction de toutes les actions dont les préconditions sont vérifiées dans la liste des fluents du niveau k.

- en addition aux actions du problème, les actions de persistance appelées les No-Ops (dont l'ensemble des préconditions coïncide avec celui des effets) sont introduites afin de

générer la liste des fluents du niveau k+1 en unissant les effets de toutes les actions introduites.

De cette manière, le graphe de planification maintient les liens de dépendance entre les actions du niveau k+1, leurs préconditions dans la liste des fluents du niveau k et leurs effets dans la liste de fluents du niveau k+1.

L'exclusion mutuelle (mutex) entre actions ou fluents peut être détectée en se basant sur les règles suivantes :

- pour un niveau donné, deux fluents sont mutex si toutes les actions de ce niveau ayant l'un comme ajout sont en exclusion mutuelle avec toutes les actions ayant l'autre fluent comme ajout.

- pour un niveau k+1 donné, deux actions sont mutex si elles interfèrent (l'ensemble de leurs préconditions est inconsistant avec celui de leurs effets) ou si elles ont des préconditions inconsistantes (une des préconditions de la première action est en exclusion mutuelle avec l'une des préconditions de la deuxième) ou si elles ont des effets inconsistants (une des actions retire un fluent se trouvant dans la liste des ajouts de l'autre action).

Chaque contrainte d'exclusion mutuelle est ensuite propagée aux niveaux suivants interdisant l'existence de deux fluents mutex dans le même état ou de deux actions à exécuter à partir du même état.

3.3.2. Phase de recherche d'un plan solution

Le but de cette phase est de vérifier s'il existe un sous-graphe dans le graphe de planification correspondant à un plan solution. La recherche se fait donc par régression en partant des fluents constituant le but dans la liste des fluents d'un niveau donné (si tous les fluents recherchés ne sont pas présents ou s'il en existe certains qui sont marqués comme mutex, on revient à la première phase en essayant de générer un niveau supplémentaire) et pour chacun de ces fluents, on sélectionne dans la liste des actions du niveau concerné celles qui le génèrent en respectant la contrainte indiquant que deux actions générant deux fluents différents ne doivent pas être sélectionnées dans le même niveau si elles sont mutex.

La recherche continue ainsi d'une manière récursive selon le même processus en choisissant comme but pour le niveau k-1 la sous-liste des préconditions des actions sélectionnées dans le niveau k.

La recherche s'arrête avec succès si on atteint le niveau 0 correspondant à l'état initial, et dans le cas contraire, on reprend l'expansion du graphe de planification par le développement d'autres niveaux, et pour chaque niveau ajouté, on reprend le processus de recherche d'un plan solution à partir du dernier niveau développé jusqu'à ce qu'on atteigne la stabilisation du graphe de planification : c'est un phénomène qu'on rencontre en tombant dans un cas où à partir d'un niveau n, tous les niveaux m tels que m>n possèdent le même ensemble de fluents, le même ensemble d'actions et les mêmes mutex sur les fluents et sur les actions.

Si on atteint la stabilisation du graphe de planification sans trouver de plan valide, on arrête le processus en concluant que le problème n'admet pas de plan valide.

Toutes les variantes de cette approche fonctionnent de la même manière que celle-ci, mais la différence entre elles réside dans la façon par laquelle elles traitent les deux phases. On peut citer par exemple :

- le planificateur IPP [Nebel et al. 1997] qui utilise plusieurs fonctions heuristiques pour effectuer une estimation de l'utilité des atomes propositionnels formant l'état initial pour résoudre l'état but. Les fluents qui sont estimés inutiles sont alors supprimés. Contrairement à la version originale du planificateur Graphplan, le planificateur IPP utilise une méthode incomplète mais il donne des résultats intéressants.

- le planificateur STAN (STAte ANalysis) [Long et Fox 1999] utilise une technique basée sur la détection des objets symétriques en s'intéressant à leur rôle dans l'état initial et le but du problème (les objets symétriques ont des rôles identiques). Il a obtenu des résultats intéressants notamment pour les problèmes ayant des domaines triviaux.

- le planificateur LCGP (Least Committed GraphPlan) [Cayrol et al. 2000 ; Cayrol et al. 2001] est une autre variante de Graphplan qui applique certaines techniques du moindre engagement.

3.4. Planification d'ordre partiel

Contrairement aux plans linéaires, les plans partiels s'intéressent à la manipulation des contraintes partielles de causalité entre les actions (si l'action α produit un fluent nécessaire pour que l'action β soit exécutée, il y a une contrainte de causalité reliant α et β).

Un planificateur d'ordre partiel [Weld 1994] est un planificateur complet basé sur le principe du moindre engagement qui repose sur l'idée de ne pas faire de choix tant qu'on n'en a pas besoin, ce qui a pour avantage d'éviter l'exécution d'un travail de parcours et de recherche qui peut être annulé ensuite si le choix sur lequel ce travail est basé se révèle incorrect. En planification, ce principe est appliqué dans le fait d'éviter d'ordonner les actions du plan partiel tant que le besoin ne s'est pas présenté. Les planificateurs d'ordre partiel utilisent ce principe : les contraintes qui ordonnent les actions (contraintes de causalité) ne sont insérées que lorsque c'est nécessaire.

Le plan partiel (ou plan partiellement construit) est représenté par un graphe décrivant les contraintes de causalité entre les actions sélectionnées. Chaque nœud dans ce graphe représente un opérateur et chaque arc représente une contrainte de causalité entre les deux nœuds reliés, une contrainte d'ordre ou une instanciation de variables.

La création d'un plan partiel se fait par une recherche dans l'espace des plans partiels dans lequel chaque nœud représente un plan partiel et chaque arc représente une transformation permettant de passer d'un plan partiel à un autre. Dans la plupart des cas, les plans partiels trouvés sont partiellement ordonnés mais il existe des cas particuliers où les plans partiellement élaborés sont des plans totalement ordonnés. On peut ainsi noter l'exemple des cubes sur une table qu'on peut déplacer en utilisant une seule pince, les plans sont alors totalement ordonnés.

Plusieurs planificateurs se basent sur la planification d'ordre partiel parmi lesquels SNLP, UCPOP [Penberthy et Weld 1992].

3.5. Planification par satisfaction de formules propositionnelles

3.5.1. Le planificateur SATPLAN

Kautz et Selman [Kautz et Selman 1992] ont proposé l'approche SATPLAN comme alternative aux approches classiques de planification dont les résultats étaient assez peu convaincants. Cette approche est basée sur la possibilité de transformer un problème de planification en un problème SAT (problème de satisfaction des formules propositionnelles).

Il est en effet possible d'effectuer cette transformation à condition que l'horizon de planification soit borné, on appelle cette approche planification par satisfiabilité.

Chaque état est décrit sous forme de conjonction de littéraux où chaque variable est indicée d'un numéro représentant l'instant de l'état en question. Par exemple, pour l'état initial s_0, toutes les variables sont indicées par 0. De même, chaque action doit être indicée par un numéro représentant l'instant auquel elle est exécutée.

Grâce à cette transformation par le biais d'un codage donné, le problème de recherche et d'extraction d'un plan valide sous horizon borné devient un problème SAT classique (dans lequel on calcule un modèle de la formule donnée quand elle est satisfaisable). Il peut alors être résolu par l'application des techniques de recherche SAT sur la base de clauses obtenue. Le modèle trouvé est traduit par un codage inverse en un plan solution.

3.5.2. Le planificateur Blackbox

Le planificateur Blackbox [Kautz et Selman 1999] peut être vu comme une extension de SATPLAN combinant plusieurs solveurs SAT (Walksat [Cohen et al. 1994] pour la recherche locale, Satz [Anbulagan et Li 1997] et Rel-sat [Bayardo et Schrag 1998] pour la recherche systématique) avec le moteur utilisé par la version originale du planificateur Graphplan [Blum et Furst 1995] dont le graphe de planification est traduit en formules logiques sous forme CNF. Cette combinaison permet d'effectuer des recherches utilisant plusieurs méthodes de parcours différentes pour garantir son efficacité à résoudre des problèmes différents. On peut ainsi paramétrer l'exécution des différents mécanismes de recherche disponibles : la recherche dans le graphe de planification avec Graphplan, la recherche locale avec Walksat, la recherche systématique par progression avec Satz et la recherche par retours-arrière intelligents avec Rel-sat.

Le système Blackbox a été lui aussi amélioré par l'utilisation d'autres techniques de recherche SAT plus performantes qui ont été exploitées dans le planificateur SatPlan-04 [Kautz 2004] ou encore dans le planificateur SatPlan-06 [Kautz et al. 2006]. Celui-ci utilise un autre codage SAT des problèmes de planification décrit dans [Kautz et al. 1996]. Il exploite aussi une nouvelle méthode pour la propagation des règles d'exclusion mutuelle (qui concerne seulement l'exclusion mutuelle entre fluents) entre les actions dans la construction du graphe de planification, ce qui a permis d'améliorer les performances du planificateur.

3.5.3. Le planificateur MaxPlan

Le planificateur MaxPlan [Xing et al. 2006] est un planificateur optimal traitant les problèmes de planification classique codés dans le formalisme STRIPS. Le critère d'optimisation est le nombre d'étapes composant le plan solution.

Dans le but de réduire la taille de l'espace de recherche à visiter, ce planificateur procède à la décomposition du problème de planification en sous-problèmes codés en formules propositionnelles afin d'obtenir un sous-problème pour chaque sous-but.

Le processus de recherche du planificateur MaxPlan commence par l'estimation d'un majorant pour la longueur du plan. Il code ensuite, pour la longueur obtenue de plan, le problème de planification en formules propositionnelles pour le résoudre grâce à un solveur SAT utilisant des techniques d'élagage et exploitant la décomposition du problème indiquée ci-dessus. Ce solveur utilise en plus des techniques d'apprentissage exploitant les similarités entre les instances SAT pour accélérer le processus de recherche. Enfin, si la recherche réussit, il enregistre la solution trouvée et il recommence le processus en diminuant la valeur du majorant, et si elle échoue, il cherche la dernière solution enregistrée qui sera la solution optimale. Si aucune solution n'est enregistrée, il recommence le processus en augmentant le majorant de la longueur du plan

Ce planificateur a obtenu des résultats très intéressants notamment dans la compétition internationale des planificateurs IPC5 en 2006.

3.6. Approche basée sur la satisfaction de contraintes

Les problèmes de satisfaction de contraintes (Constraint Satisfaction Problems ou CSP) permettent de modéliser, étudier et résoudre de nombreux problèmes d'intelligence artificielle et de recherche opérationnelle grâce à la grande expressivité offerte par la notion de contraintes.

Un CSP est un triplet $(\mathcal{V}, \mathcal{D}, \mathcal{C})$ formé par l'ensemble $\mathcal{V}$ des variables qui ont leurs domaines respectifs de valeurs dans l'ensemble $\mathcal{D}$ de domaines (des ensembles finis). L'ensemble $\mathcal{C}$ est l'ensemble des contraintes qui traduisent les relations entre les différentes variables du CSP.

Résoudre un CSP est un problème qui consiste à affecter une valeur à chacune de ses variables dans son domaine en respectant toutes les contraintes du problème. Si une telle solution existe alors le CSP est consistant. C'est un problème $\mathcal{NP}$-difficile [Garey et Johnson 1979] tout comme le problème SAT qui constitue un cas particulier du problème CSP à domaines finis.

Vu la complexité du problème CSP, une grande importance est donnée aux techniques de réduction de l'espace de recherche appelées aussi techniques de filtrage qui ont pour but de réduire l'espace de recherche en éliminant de chaque domaine des variables les valeurs impossibles, c'est-à-dire celles dont on peut prouver la non-appartenance à aucune des solutions du CSP.

Les principales techniques de filtrage sont basées sur la consistance locale qui permet de vérifier, pour une variable donnée, s'il y a des valeurs pour les autres variables qui violent les contraintes qui lui sont liées. Ces techniques, qu'on peut appliquer avant ou pendant la phase de recherche de solution, cherchent à éliminer les éléments qui ne peuvent pas localement être présents dans une solution.

Parmi les techniques de consistance locale il y a la consistance d'arc (ou arc-consistance) [Waltz 1975] qui s'intéresse à vérifier la consistance locale pour les CSP binaires (les CSP où chaque contrainte met en relation deux variables). Cette technique peut être généralisée pour les contraintes à n variables et elle est appelée dans ce cas GAC [Bessière et Régin, 1997]. La consistance locale peut être appliquée aux variables à domaines continus en s'intéressant aux bornes inférieure et supérieure. La technique utilisée est la 2B-consistance [Lhomme 1993].

Il existe plusieurs autres consistances locales parmi lesquelles on peut citer la k-consistance qui est une consistance locale qui assure que l'on peut étendre toute instanciation partielle consistante de k-1 variables à k variables [Freuder 1978].

Pour répandre le filtrage par consistance locale sur tout le CSP, celle-ci est appliquée d'une façon récursive par les techniques de propagation des contraintes jusqu'à ce qu'aucune nouvelle information ne puisse être déduite. On obtient alors la fermeture par *P*-consistance du CSP où *P* représente la technique de consistance locale considérée.

Un problème de satisfaction de contraintes dynamique ou DCSP [Mittal et Falkenhainer 1990] est une généralisation du CSP pour les contraintes conditionnelles dans laquelle seul un

sous-ensemble des variables est initialement pris en considération (ces variables sont alors mentionnées comme activées) et l'objectif est de trouver une assignation de toutes les variables activées qui soit consistante avec les contraintes qui les concernent.

En plus des composantes du CSP, un DCSP contient un ensemble de contraintes appelées contraintes d'activité qui sont des contraintes conditionnelles exprimant le fait suivant : « Si la variable x a pour valeur v_x, alors les variables y,z,… sont activées ».

Dans l'algorithme de planification selon le mécanisme Graphplan, le graphe de planification peut être vu comme un DCSP [Kambhampati 2000]. Une transition du graphe de planification en DCSP est possible en procédant comme suit :

- les fluents des différents niveaux du graphe sont considérés comme des variables du DCSP.
- les contraintes d'exclusion mutuelle entre fluents sont exprimées de la manière suivante : pour deux fluents f_1 et f_2 en mutex, on a $\neg(\text{Active}(f_1) \wedge \text{Active}(f_2))$
- les contraintes d'exclusion mutuelle entre actions sont exprimées par rapport aux fluents produits par ces actions : si deux actions α_1 et α_2 sont mutex, alors pour chaque couple de fluents (f_1, f_2) pour lequel α_1 est un support possible pour f_1 (c'est-à-dire α_1 peut générer le fluent f1) et α_2 est un support possible de f_2, nous avons la contrainte $\neg(f_1 = \alpha_1 \wedge f_2 = \alpha_2)$
- les contraintes d'activation des sous-buts sont exprimées par les préconditions des actions : si f est un fluent actif généré par une action α, alors tous les fluents du niveau précédent qui correspondent aux préconditions de α sont activés.
- les fluents correspondant au but du problème sont activés dès le début.

L'intérêt de la transformation du problème de planification en un DCSP est de bénéficier des différents mécanismes puissants de résolution des CSP (en particulier les mécanismes de filtrage par consistance locale) pour résoudre les problèmes de planification. Il est en effet possible de transformer un DCSP en un CSP en ajoutant la valeur NULL dans les domaines de chaque variable du problème. Cette valeur sera affectée aux variables inactives. En plus, le DCSP possède ses propres méthodes de résolution comme la méthode de recherche par progression dans laquelle, pour chaque niveau, on affecte aux variables activées des valeurs respectant les différentes contraintes en commençant par le niveau 0 correspondant à l'état initial.

Le processus s'arrête avec succès si on arrive à affecter à toutes les variables activées du problème des valeurs consistantes avec toutes les contraintes après avoir atteint un état dans lequel aucune variable ne peut plus être activée. Il détecte un échec s'il y a des variables activées pour lesquelles il n'y a aucune affectation possible consistante avec les contraintes et dans ce cas on doit effectuer un retour-arrière (backtrack) et réessayer d'autres affectations.

Parmi les planificateurs qui utilisent les techniques de résolution des CSP, on peut citer les planificateurs CPlan [Chen et Van Beek 1999] et GP-CSP [Kambhampati et Do 2000].

CONCLUSION

En planification classique, plusieurs restrictions ont été admises pour limiter l'espace d'états et rendre possible la recherche de plans valides. Ces restrictions rendent difficiles la prise en compte des problèmes du monde réel mais elles étaient nécessaires pour que les premiers planificateurs puissent fournir des résultats en un temps raisonnable.

Avec les progrès réalisés en intelligence artificielle et en recherche opérationnelle, les planificateurs ont acquis de grandes performances et sont désormais capables de traiter des problèmes de plus en plus difficiles grâce aux nouveaux algorithmes de recherche.

Pour cette raison, la résolution de nouvelles classes de planification qui sont plus générales a été rendue envisageable comme par exemple la planification temporelle ou encore la planification dans l'incertain. Dans ces classes de planification, certaines restrictions de la planification classique au sens strict ont été omises.

Dans le prochain chapitre, nous allons étudier les extensions de la planification classique introduisant le temps et les ressources permettant ainsi de traiter des problèmes encore plus proches du monde réel et de définir d'autres critères pour caractériser les plans optimaux.

CHAPITRE 2

EXTENSIONS DE LA PLANIFICATION POUR LA GESTION DU TEMPS ET DES RESSOURCES

INTRODUCTION

Au chapitre précédent, on a pu constater que la planification classique exige des restrictions qui la rendent peu expressive pour représenter les problèmes du monde réel. Pour cette raison, plusieurs extensions ont été introduites, comme la planification temporelle, la planification avec des variables et contraintes numériques, la planification dans un environnement incertain,…

Dans ce chapitre, nous allons présenter différents travaux réalisés en planification temporelle, notamment en ce qui concerne les formalismes de représentation des données temporelles et les planificateurs temporels les plus remarquables.

En plus, nous allons étudier certains formalismes utilisés pour la représentation des variables et des contraintes numériques (notamment les ressources) dans les problèmes d'ordonnancement et de planification temporelle.

1. PLANIFICATION TEMPORELLE

1.1. La planification temporelle comme extension de la planification classique

La planification temporelle est une extension de la planification classique obtenue en supprimant certaines restrictions et en ajoutant certaines notions :

- les actions en planification temporelle ont des durées distinctes, contrairement à la planification classique où toutes les actions sont équivalentes sur le point de vue temporel.

- il est possible d'exécuter plusieurs actions simultanément si elles ne sont pas mutuellement exclusives. Cette propriété existe déjà dans certains aspects de la planification classique comme la planification Graphplan ou la planification par moindre engagement, mais l'apport de la planification temporelle réside dans le fait de pouvoir exécuter plusieurs actions successives de « courtes durées » pendant l'exécution d'une action de « longue durée ».

- en plus de la notion de plan valide (qui permet d'atteindre le but), il y a la notion de plan optimal. Celle-ci a subi certains changements entre la planification classique et la

planification temporelle; en effet, un plan optimal dans le cadre classique est un plan réalisable qui permet d'atteindre le but en exécutant un nombre minimum d'actions, tandis qu'en planification temporelle, un plan optimal est un plan réalisable qui permet d'atteindre le but en minimisant le temps total d'exécution entre l'état initial et l'état but.

La planification temporelle permet aussi de gérer les ressources limitées, mais les résultats réalisés dans ce domaine par les planificateurs temporels actuels sont encore insuffisants soit au niveau du temps de recherche d'un plan valide (ou optimal), soit au niveau des restrictions admises par les planificateurs pour représenter les données numériques et qui éloignent les problèmes de planification traités du monde réel.

1.2. Formalismes de représentation du temps dans la planification temporelle

Le facteur le plus influent dans le raisonnement temporel pour la planification est le formalisme dans lequel le temps est modélisé pour les actions et les fluents.

La plupart des planificateurs temporels actuels utilisent l'un des trois formalismes suivants :

- le formalisme basé sur les treillis d'instants [Ghallab et al. 1988].
- le formalisme basé sur l'exclusion mutuelle [Smith et Weld 1999].
- le formalisme PDDL introduit dans PDDL2.1 [Fox et Long 2001] et PDDL+ [Fox et Long 2002].

1.2.1. Formalisme basé sur les treillis d'instants

Les treillis d'instants constituent le cadre temporel utilisé par le planificateur IxTeT [Ghallab et al. 1988]. Ils prennent en compte les contraintes et les relations entre instants ou entre intervalles de temps avec des actions ayant une durée.

Dans un treillis d'instants $(\mathcal{T}, \leq)$, le temps n'est pas continu ; il est discrétisé avec un pas (différence de temps entre deux instants successifs) adapté pour qu'il soit suffisant pour représenter les évènements du monde réel. La relation entre les instants est une relation

d'ordre partiel $\leq$ permettant d'indexer l'ensemble $\mathcal{T}$ représentant le temps alors que les contraintes temporelles sont représentées par une carte temporelle qui consiste en un graphe de contraintes temporelles numériques (contraintes de durées et de précédence) sur l'ensemble des instants.

Une table de temps discrète *OTT* (Operator Time Table) est associée à chaque opérateur dans laquelle l'ensemble des actions, propriétés et effets liés à l'opérateur est situé.

En planification temporelle, pour insérer un opérateur dans le plan en cours de formalisation, il faut placer les éléments de l'*OTT* associée à cet opérateur par rapport aux instants déjà inclus dans le plan tout en préservant la consistance de celui-ci avec les différentes contraintes du problème.

Les propositions temporelles dans ce formalisme se basent sur les assertions qui expriment respectivement le maintien de la valeur d'un opérateur sur intervalle de temps (*hold*) et les évènements qui expriment les transitions sur les valeurs des opérateurs à des instants donnés (*event*).

1.2.2. Formalisme basé sur l'exclusion mutuelle

Le succès que les planificateurs de type Graphplan ont connu pour le cadre classique de la planification a incité Smith et Weld à étendre le raisonnement suivi par ces algorithmes au cadre temporel dans le planificateur TGP qu'ils ont développé dans [Smith et Weld 1999]. Ils ont ainsi adopté une simple extension du formalisme STRIPS dans laquelle un problème de planification temporelle est codé par un n-uplet ($\mathcal{A}$, $\mathcal{S}_0$, O, $\mathcal{G}$) où :

- $\mathcal{A}$ est l'ensemble des atomes de base.
- $\mathcal{S}_0 \subseteq \mathcal{A}$ et $\mathcal{G} \subseteq \mathcal{A}$ représentent respectivement l'état initial et l'état but du problème.
- O est l'ensemble des opérateurs STRIPS (totalement instanciés) de base.

Pour chaque opérateur (action) $\alpha \in O$:

- il existe un instant de départ T(α) et une durée $dur(\alpha)$ relatifs à l'action α.
- toutes les préconditions de l'action α doivent être vérifiées à l'instant T(α).

- les préconditions de l'action α qui ne sont pas affectées (modifiées) par celle-ci doivent rester vérifiées durant toute l'exécution de α, c'est-à-dire durant l'intervalle de temps [T(α),T(α)+*dur*(α)].

- la situation des effets de l'action α reste indéfinie durant l'exécution de celle-ci et tous les effets sont vérifiés à l'instant où α achève son exécution, c'est-à-dire l'instant T(α)+*dur*(α).

Ces règles expliquent que deux actions α_1 et α_2 ne peuvent pas s'exécuter en parallèle si un effet ou une précondition de l'une est la négation d'un effet ou d'une précondition de l'autre. Dans ces cas, les deux actions α_1 et α_2 sont alors en exclusion mutuelle (mutex).

L'exclusion mutuelle introduite dans Graphplan est généralisée par la notion de l'exclusion mutuelle éternelle (e-mutex) pour le cadre temporel par les règles suivantes :

- deux fluents sont en exclusion mutuelle éternelle si et seulement si l'un est la négation de l'autre.

- une action α est en exclusion mutuelle éternelle avec un fluent f si la négation de celui-ci (¬f) est une précondition ou un effet de α ou si f est un effet de α.

- deux actions α_1 et α_2 sont en exclusion mutuelle éternelle si l'une d'elles supprime une précondition ou un effet de l'autre ou si leurs préconditions sont en mutex.

Dans le cas d'une exclusion mutuelle (éternelle ou non) entre deux actions α et β, il n'existe aucune unité de temps t_i pendant laquelle ces deux actions s'exécutent simultanément. Autrement dit, ces deux actions doivent être organisées de manière à ce que l'une des deux doive commencer son exécution après que l'autre termine la sienne.

Le planificateur TGP (Temporal GraphPlan) tire profit de ce formalisme pour adapter la construction du graphe de planification au cadre de planification temporelle. Il procède ainsi à la représentation du graphe de planification en utilisant des intervalles de temps à la place des niveaux d'actions parallèles et vérifier, pour chaque intervalle, les actions à y exécuter en parallèle. En plus, il utilise l'extension de la notion d'exclusion mutuelle afin de détecter les menaces entre des actions ayant des durées différentes.

1.2.3. Formalisme PDDL

Le langage PDDL est utilisé essentiellement pour la planification classique, il a été étendu par PDDL2.1 [Fox et Long 2001] utilisé pour la première fois à la compétition IPC3 en 2002. Cette version permet de modéliser les problèmes de planification temporelle.

a. PDDL2.1

Pour décrire les données des problèmes de planification temporelle, Maria Fox et Derek Lang ont développé PDDL2.1 [Fox et Long 2001], une version améliorée de PDDL en intégrant entre autres, la durée des actions et les contraintes numériques.

Un opérateur PDDL temporel est un quadruplet op = ($\mathcal{P}rec$(op), add(op), del(op), dur(op)) où $prec$(op), add(op) et del(op) sont des atomes de base qui dénotent respectivement les préconditions, ajouts et retraits de op, et dur(op) est un nombre rationnel qui représente la durée de op.

Le langage PDDL2.1 est décomposé en 4 niveaux d'expressivité :

Premier niveau

Le premier niveau représente un enrichissement du formalisme STRIPS se manifestant particulièrement dans :

- la décomposition de l'ensemble des préconditions en deux sous-ensembles, un pour les fluents affirmatifs (ayant la valeur *VRAI*) et un autre pour les fluents négatifs (ayant la valeur *FAUX*).
- l'expression des effets conditionnels.
- l'expression des préconditions disjonctives.
- la possibilité d'utilisation des quantificateurs existentiels et universels.

Ce niveau offre donc à PDDL le même pouvoir d'expressivité que celui du langage ADL élaboré par Pednault [Pednault 1989].

Deuxième niveau

Le deuxième niveau introduit les variables numériques (utilisées essentiellement pour représenter les ressources) avec la possibilité de tester et de changer instantanément leur valeur.

Troisième niveau

Le troisième niveau permet une représentation simplifiée des problèmes de planification temporelle en offrant les possibilités suivantes :

- la représentation des actions ayant une durée propre.
- la description des cas où il est possible (ou impossible), pour certaines actions, d'être exécutées simultanément avec d'autres actions.
- la description des effets temporels, c'est-à-dire les effets de début qui sont valides à l'instant où l'action concernée commence son exécution et les effets de fin qui sont valides à l'instant où l'action termine son exécution.
- la description des conditions temporelles, c'est-à-dire les préconditions de début qui doivent être vérifiées à l'instant où l'action concernée commence son exécution, les conditions de fin qui doivent être vérifiées à l'instant où l'action termine son exécution et les conditions invariantes qui doivent rester vérifiées tout le long de l'exécution de l'action concernée.

Il est à noter que dans le niveau 3, la durée d'une action peut aussi être exprimée en fonction d'autres variables (numériques ou pas) du problème.

Quatrième niveau

Le quatrième niveau est une extension du troisième niveau en ajoutant la possibilité d'expression des effets continus, c'est-à-dire d'exprimer des effets qui changent durant l'exécution d'une autre action donnée, par exemple le changement des valeurs liées à une ressource numérique continuement tout au long de l'exécution d'une action.

Dans PDDL2.1, il y a aussi la possibilité d'exprimer des effets « indépendants », c'est-à-dire des actions reliées à des temps spécifiques pour lesquels le planificateur n'a pas de contrôle (par exemple le lever / le coucher du soleil) et la possibilité d'exprimer un temps-limite (deadline).

Enfin, il faut noter que PDDL 2.1 permet de définir un critère de préférence caractérisant l'optimalité d'un plan valide, qui peut être le temps de son exécution ou la quantité consommée d'une ressource donnée durant son exécution (par exemple le carburant) ou une fonction prenant en compte des paramètres temporels et numériques.

b. PDDL+

La subtilité majeure qui a nécessité l'apparition de PDDL+ [Fox et Long 2002] est liée à l'introduction du temps. En effet, dans la planification classique, seul l'ordre des actions importe, tandis qu'en planification temporelle, l'important est le temps de commencement de l'exécution de l'action et le temps de sa fin. Il y a de ce fait une grande importance donnée à l'instant exact où une précondition devient vérifiée et l'instant exact où un effet s'accomplit. Mais il n'est pas demandé de préciser, pour la validité d'un plan, si une action α, qui génère un effet nécessaire pour l'exécution d'une autre action β, doit terminer son exécution avant que β ne commence la sienne.

Ce problème se manifeste lorsque, par exemple, deux actions dont les préconditions de l'une sont les effets de début de l'autre commencent leurs exécutions, ce qui implique qu'on ne peut pas dire que ces deux actions doivent s'exécuter « exactement » au même instant.

Pour résoudre ce problème, PDDL+ a introduit dans son cinquième niveau d'expressivité la notion de « valeur de tolérance » représentée par le temps « ε » : c'est la plus grande valeur d'exactitude qu'on peut générer. C'est-à-dire que si une précondition est dépendante d'un effet, ε est considérée comme la plus petite taille pour l'intervalle de temps, entre les deux évènements, suffisante à garantir le fait que l'effet va être accompli et activé avant de tester la précondition. Sans cet intervalle, on ne peut pas garantir ces évènements, ce qui pourrait générer des plans invalides dans le monde réel.

1.3. Exemples de planificateurs temporels

Le formalisme PDDL est bien plus riche que celui basé sur l'exclusion mutuelle. Cependant, malgré la capacité des planificateurs temporels actuels à lire les problèmes temporels codés en PDDL2.1 ou PDDL+, on constate que la plupart d'entre eux demeurent incapables d'exploiter tous les apports de ce langage ; rares sont ceux qui arrivent à exploiter

le quatrième ou le cinquième niveau d'expressivité, et même pour le troisième niveau, plusieurs planificateurs temporels se limitent au cadre d'expressivité précisé par le formalisme basé sur l'exclusion mutuelle de Smith et Weld.

Le formalisme basé sur les treillis d'instants se révèle plus expressif que PDDL+ dans plusieurs points. Cependant, à notre connaissance, seul le planificateur IxTeT [Ghallab et Laborie 1995] profite de toute l'expressivité de ce formalisme.

Nous allons présenter ici quelques planificateurs temporels récents utilisant des adaptations des mécanismes de recherche de plans valides pour le cadre temporel. Ces planificateurs sont : IxTeT [Ghallab et Laborie 1995], LPG [Gerevini et al. 2005], SAPA [Kambhampati et Do 2001 ; Kambhampati et Do 2003], TP4 [Geffner et Haslum 2001] et CPT [Geffner et Vidal 2004 ; Geffner et Vidal 2005].

1.3.1. Le planificateur IxTeT

IxTeT (IndeXed TimE Table) [Ghallab et Laborie 1995] est un planificateur utilisant une adaptation du principe du moindre engagement pour le cadre temporel afin de produire des plans partiels. Il utilise, en plus, certaines heuristiques pour guider sa recherche dans une structure hiérarchique d'actions.

Le domaine de planification est représenté par des littéraux positifs codés selon le formalisme STRIPS. En plus, IxTeT utilise une table à deux dimensions dont une représente le temps, selon le formalisme basé sur les treillis d'instants et l'autre représente les attributs et les ressources du domaine.

Dans le système hiérarchique des actions, celles-ci sont transformées en tâches (opérateurs de planification) qui peuvent elles-aussi utiliser des sous-tâches définies de la même manière à condition que le réseau hiérarchique obtenu soit sans circuit.

L'état initial est représenté par une tâche particulière qui contient un ensemble d'évènements donnant aux attributs leur valeur initiale, un ensemble d'effets et le niveau initial des ressources. Les buts sont représentés par des assertions ou des évènements.

La recherche d'un plan partiel commence par l'état initial qui est considéré comme plan partiel avec des « failles » éventuelles. Les branches représentent des tâches ou des contraintes introduites dans le plan courant pour résoudre une faille.

Les failles peuvent être classées en plusieurs catégories :

- des buts ou des sous-buts pour lesquels les préconditions ne sont encore par établies. Dans ce cas, elles peuvent être résolues par l'ajout de liens causaux ou de nouvelles instances d'actions.

- des menaces, qui sont des évènements ou des assertions menaçant un autre lien causal déjà en place. Dans ce cas, elles peuvent être résolues en les ordonnant par des méthodes constructives ou destructives, par l'ajout d'une contrainte d'inégalité ou par l'ajout de tâches produisant la ressource.

Cette recherche est guidée par des heuristiques offrant un bon compromis entre le temps de recherche et la qualité du plan valide trouvé (le temps consacré à son exécution), ce qui veut dire que IxTeT n'est pas optimal mais il peut générer un plan optimal en suivant le processus suivant :

- donner des temps-limites aux différentes tâches.

- puisque le temps est représenté d'une manière discrète, décrémenter les temps-limites donnés par un pas de discrétisation jusqu'à ce qu'aucun plan valide ne soit trouvé. Le dernier plan valide trouvé est donc le plan optimal.

1.3.2. Le planificateur LPG

LPG (Local search for Planning Graphs) [Gerevini et Serina 2002 ; Gerevini et al. 2005] est un planificateur temporel utilisant une extension du planificateur Graphplan pour le cadre numérique et temporel. La phase d'extraction du plan solution utilise un mécanisme de recherche locale basée sur une démarche inspirée du solveur SAT Walksat [Cohen et al. 1994] permettant ainsi de résoudre des problèmes d'adaptation et de génération de plans.

Le planificateur LPG est capable de lire les problèmes codés en PDDL2.1, cependant, il utilise le formalisme de Smith et Weld basé sur l'exclusion mutuelle pour représenter les données temporelles.

L'espace de recherche du planificateur LPG se compose des graphes d'actions : des sous-graphes particuliers du graphe de planification représentant des plans partiels. Les étapes de recherche sont constituées de certaines modifications du graphe transformant un graphe d'actions en d'autres. Le planificateur LPG exploite une représentation concise du graphe de planification pour définir le voisinage de recherche et évaluer ses éléments en utilisant une fonction paramétrée, dont les paramètres pèsent différents types d'incohérences dans le plan partiel actuel et sont dynamiquement évaluées pendant la recherche.

La fonction d'évaluation utilise des heuristiques pour l'estimation du coût de recherche et du coût d'exécution nécessaires pour réaliser une précondition (qui peut être numérique). Les durées et les quantités numériques (la consommation de carburant par exemple) liées aux actions sont représentées dans les graphes d'actions et sont modélisées dans la fonction d'évaluation. Dans les domaines temporels, les actions sont triées en utilisant un graphe de précédence qui est mis à jour pendant la recherche et qui tient compte des relations d'exclusion mutuelle du graphe de planification.

Le système peut produire des plans de bonne qualité avec un ou plusieurs critères d'optimisation. Ceci est réalisé par un processus produisant une séquence de plans, dont chacun contient une amélioration de la qualité du précédent.

1.3.3. Le planificateur SAPA

SAPA [Kambhampati et Do 2001 ; Kambhampati et Do 2003] est un planificateur qui utilise le chaînage avant basé sur des heuristiques pouvant traiter des actions temporelles (durables) et des contraintes numériques de ressources. Il est capable de lire les problèmes codés en PDDL2.1, cependant, il utilise un formalisme propre de représentation du cadre temporel avec ressources qui est très semblable au quatrième niveau d'expressivité de PDDL2.1 avec la représentation des buts avec date-limite (deadline) mais il offre une représentation plus étendue des actions mutex.

Les développeurs du planificateur SAPA se sont particulièrement concentrés sur le problème de développement d'heuristiques liées aux problèmes de pertinence et de coût. Ils ont développé une approche de planification basée sur les graphes pour réaliser cet objectif. Cette approche consiste à évaluer (en fonction du temps) la structure de données du graphe de

planification par un mécanisme d'estimation du coût d'exécution des buts et des sous-buts pour utiliser ensuite l'algorithme A* de recherche de plan. Ils ont ainsi implémenté une méthodologie pour estimer les fonctions de coût, qui peut être utilisée comme base pour dériver l'heuristique afin de supporter toute fonction basée sur la distance jusqu'au but et le coût d'exécution. Cette heuristique est basée sur l'évaluation du graphe relaxé de planification (RTPG).

1.3.4. Le planificateur TP4

Le planificateur TP4 [Geffner et Haslum 2001] est capable de lire des problèmes de planification codés en PDDL2.1. Cependant, il utilise le formalisme de Smith et Weld basé sur l'exclusion mutuelle pour représenter les données temporelles. Il est à signaler que la représentation du temps dans TP4 se fait par des rationnels positifs (même si, dans la plupart des cas, le temps sera représenté par des entiers).

Ce planificateur effectue une recherche en utilisant la régression temporelle, qui est un mécanisme basé sur un enchaînement régressif dans l'espace des « queues des plans ».

Dans la planification classique, les queues de plans (en anglais « plan tails ») sont des plans partiels construits à partir de l'action finale, et quand une queue de plan est concaténée avec une tête de plan, le résultat est un plan valide. Une queue de plan peut être représentée par un ensemble de littéraux et ce sont ces ensembles qui forment les états dans l'espace de recherche.

Cependant, dans la planification temporelle, il est possible d'avoir une action encore en cours d'exécution à l'instant où un état est atteint. Il est donc nécessaire d'enrichir la notion d'état pour qu'elle tienne en compte ce paramètre. La représentation d'un état dans le cadre temporel est alors assurée par un couple $(\mathcal{E},\mathcal{F})$ pour lequel $\mathcal{E}$ est l'ensemble des littéraux vérifiés (ayant la valeur *TRUE*) à l'instant où l'état est vérifié et $\mathcal{F}=\{(\alpha_1,\delta_1),\ldots,(\alpha_n,\delta_n)\}$ est l'ensemble des actions représentées dans le temps par des couples (α_i,δ_i). Ceci veut dire qu'un plan $\mathcal{P}$ conduit à l'état $\mathcal{S}_i = (\mathcal{E},\mathcal{F})$ à l'instant t si $\mathcal{P}$ permet de valider tous les littéraux de l'ensemble $\mathcal{E}$ à l'instant t et pour lequel dans chaque couple $(\alpha_i,\delta_i) \in \mathcal{F}$, l'action α_i est exécutée à l'instant $t-\delta_i$.

L'état initial est $\mathcal{S}_0 = (\mathcal{G}_p,\varnothing)$ où $\mathcal{G}_p$ est l'ensemble des littéraux constituant le but et les états finaux sont de la forme $(\mathcal{E},\varnothing)$ où $\mathcal{E}$ est un sous-ensemble de l'ensemble des fluents initiaux.

Pour garantir la persistance des fluents dans le temps, TP4 utilise des actions No-Ops ayant un ensemble de préconditions qui coïncide avec celui des effets (comme dans les planificateurs classiques de style Graphplan) et qui intègrent une information supplémentaire indiquant la durée de ces actions.

Le planificateur TP4 utilise l'heuristique de calcul du coût optimal H* (étroitement lié au temps puisque c'est le critère d'optimalité dans TP4) donnée comme suit : pour un état $\mathcal{S}i=(\mathcal{E},\mathcal{F})$, le coût optimal est $H^*(\mathcal{S}_i)=t$ si et seulement si t est le dernier instant tel qu'il existe un plan $\mathcal{P}$ qui atteint et satisfait l'état $\mathcal{S}_i$ à t. Cette fonction récurrente est formulée par l'équation de Bellman :

$$H^*(\mathcal{S}_i)= \begin{cases} 0 \text{ si } \mathcal{S}i \text{ est un état but} \\ \text{Min}_{(\mathcal{S}j\in R(\mathcal{S}i))}\, C(\mathcal{S}_i,\mathcal{S}_j) + H^*(\mathcal{S}_j) \end{cases}$$

avec :

-$R(\mathcal{S}_i)$ est l'ensemble des états pouvant être construits à partir de $\mathcal{S}_i$ par régression ;

- $C(\mathcal{S}_i,\mathcal{S}_j)$ est le coût réel de passage de $\mathcal{S}_j$ à $\mathcal{S}_i$.

Cette équation ne peut pas être résolue en pratique, c'est pourquoi les développeurs de TP4 ont cherché à déterminer un minorant de H* en se basant sur le fait qu'un plan qui atteint et satisfait l'état $\mathcal{S}_i$ doit avoir satisfait dans chaque couple $(\alpha_i,\delta_i)\in\mathcal{F}$ les préconditions, ce qui donne les inégalités suivantes :

$$H^*(\mathcal{E},\mathcal{F}) \geq \text{Max}_{(\alpha k,\delta k)\in\mathcal{F}} H^*\Big(\bigcup_{(\alpha i,\delta i)\in F\ ;\delta i\geq\delta k} \text{prec}(\alpha_i),\varnothing\Big)+\delta_k.$$

$$H^*(\mathcal{E},\mathcal{F}) \geq \text{Max}_{(\alpha k,\delta k)\in\mathcal{F}} H^*\Big(\mathcal{E}\cup\bigcup_{(\alpha i,\delta i)\in F} \text{prec}(\alpha_i),\varnothing\Big).$$

Et puisque la validation de l'ensemble d'atomes E nécessite la validation de tout sous-ensemble $\mathcal{E}'\in 2^{\mathcal{E}}$ (l'ensemble des parties de $\mathcal{E}$), l'inégalité suivante doit aussi être respectée :

$$H^*(\mathcal{E},\varnothing) \geq \text{Max}_{\mathcal{E}'\subseteq\mathcal{E};|\mathcal{E}'|\leq m} H^*(\mathcal{E}',\varnothing)$$

où m est un entier positif quelconque.

Pour le cadre temporel, un minorant H_T^m de la fonction H* est défini en se basant sur les inégalités précédentes et en fixant la valeur de l'entier m :

- si l'état $\mathcal{E}$ coïncide avec l'état initial du problème, $H_T^m(\mathcal{E},\varnothing) = 0$.

- si le nombre de littéraux de $\mathcal{E}$ est inférieur ou égal à m,

$H_T^m(\mathcal{E},\varnothing) = \text{Min}_{\mathcal{S}j \in R(\mathcal{S}i=(\mathcal{E},\varnothing))} C(\mathcal{S}_i=(\mathcal{E},\varnothing),\mathcal{S}_j) + H_T^m(\mathcal{S}_j)$.

- si le nombre de littéraux de $\mathcal{E}$ est strictement supérieur à m,

$H^*(\mathcal{E},\varnothing) = \text{Max}_{\mathcal{E}' \subseteq \mathcal{E}; |\mathcal{E}'| \leq m} H^*(\mathcal{E}',\varnothing)$.

- enfin, pour le cas général,

$$H_T^m(\mathcal{E},\mathcal{F}) = \text{Max}\ [\text{Max}_{(\alpha k,\delta k)\in\mathcal{F}}\ H_T^m\Big(\bigcup_{(\alpha i,\delta i)\in F\ ;\delta i \geq \delta k} \mathcal{P}rec(\alpha_i),\varnothing) + \delta_k),$$

$$H_T^m\Big(\mathcal{E} \cup \bigcup_{(\alpha i,\delta i)\in F} \mathcal{P}rec(\alpha_i),\varnothing)\Big]$$

Pour m=1, lorsque le nombre de littéraux de $\mathcal{E}$ est inférieur ou égal à m, on a :

$H_T^1(\{p\},\varnothing) = \text{Min}_{\alpha;\, p\in} \mathcal{add}(\alpha)\ \mathcal{dur}(\alpha) + H_T^1(\mathcal{P}rec(\alpha), \varnothing)$.

En utilisant l'algorithme de recherche IDA* et l'heuristique H_T^m (avec m fixé à 1 ou 2), le planificateur TP4 permet d'obtenir un plan valide pouvant contenir des actions parallèles sous forme d'ordonnancements. Il permet aussi de trouver des plans optimaux selon le critère temporel (le meilleur plan est celui qui s'exécute en un temps minimal). Cependant, pour l'algorithme IDA*, le PGCD de deux nombres rationnels a et b est le plus grand rationnel c tel que a= m.c et b= n.c avec m et n deux entiers, ce qui permet de donner des PGCD qui peuvent être infiniment petits compliquant ainsi la démarche de l'optimisation de cet algorithme avec la régression temporelle qui utilise ce PGCD comme pas de changement pour les bornes du coût.

Pour éviter un tel problème, TP4 effectue l'optimisation en utilisant le processus suivant :

- au début, les durées des actions sont arrondies aux entiers les plus proches.

- le problème ainsi obtenu est résolu par l'algorithme IDA* et le coût calculé du plan solution (qui est aussi un plan valide du problème original) constitue un majorant du plan optimal du problème originel. Ce coût peut ne pas être optimal pour le problème original mais il est généralement assez proche.

- enfin, après avoir restauré le problème avec les durées originales des actions, le planificateur effectue une recherche avec l'algorithme Branch-and-Bound commençant par le majorant déjà trouvé dans l'étape précédente afin d'obtenir une solution optimale.

Cette méthode permet une recherche du plan optimal plus rapide que via l'application de l'algorithme IDA* directement sur le problème original car dans ce cas, l'algorithme IDA* sera appliqué à des valeurs entières dont le PGCD sera toujours au moins égal à 1.

1.3.5. Le planificateur CPT

Le planificateur CPT [Geffner et Vidal 2004 ; Geffner et Vidal 2005] est un planificateur temporel optimal basé sur la combinaison du branchement dans l'espace des plans partiels avec des liens causaux (Partial Order Causal Links ou POCL) intégrant certaines heuristiques existantes et l'utilisation de règles d'élagage basées sur la programmation et la propagation des contraintes. Cette combinaison permet de générer des plans optimaux avec la méthode branch-and-prune.

Pour représenter le cadre temporel, ce planificateur utilise le formalisme de Smith et Weld basé sur l'exclusion mutuelle et il permet ainsi de trouver des plans parallèles et optimaux (en prenant le temps comme critère d'optimisation). Il présente une innovation qui se manifeste dans l'élaboration de règles d'élagage adaptées permettant de trouver rapidement les affectations invalides et éviter ainsi des branchements dans des parties non pertinentes de l'espace de recherche.

Comme pour le planificateur TP4, le planificateur CPT utilise l'heuristique H^m_T pour effectuer une estimation de la durée nécessaire pour atteindre le but.

a. Planification POCL

Dans la planification POCL classique [McAllester et Rosenblitt 1991, Weld 1994], un plan partiel (appelé aussi état) est représenté par un n-uplet σ = (*STEPS, ORD, CL, OPEN*) où :

- *STEPS* est l'ensemble des actions du plan, complété par deux actions fictives *START* et *END* représentant respectivement le commencement et la fin du plan.
- *ORD* est l'ensemble des contraintes de précédence entre les éléments de *STEPS*.
- *CL* est l'ensemble des liens causaux (voir définition 2.1).
- *OPEN* est l'ensemble des préconditions ouvertes (i.e. non satisfaites).

Définition 2.1

Lorsque une action α_1 appartient à la liste des supports d'une précondition p de l'action α_2, une entité appelée lien causal est alors créée entre α_1 et α_2. Elle est notée $\alpha_1[p]\alpha_2$ (une autre notation de la forme $\alpha_1 \xrightarrow{p} \alpha_2$ est parfois utilisée dans la littérature).

Illustration

Dans l'exemple ZenoTravel, l'action *Fly*(a_1,c_2,c_3) permet à l'avion a_1 de voler à une vitesse normale de la ville c_2 à la ville c_3. Elle ne peut être exécutée que si l'avion a_1 se trouve à la ville c_2. C'est à dire qu'elle a pour précondition le prédicat *At*(a_1,c_2).

Les deux actions *Fly*(a_1,c_1,c_2) et *Zoom*(a_1,c_1,c_2) appartiennent à la liste des supports de cette précondition, puisqu'elles ont toutes les deux pour effet de déplacer l'avion a_1 à la ville c_2.

Pour cette raison, deux liens causaux peuvent être créés :

- *Fly*(a_1,c_1,c_2)[*At*(a_1,c_2)]*Fly*(a_1,c_2,c_3) et
- *Zoom*(a_1,c_1,c_2)[*At*(a_1,c_2)]*Fly*(a_1,c_2,c_3).

Le mécanisme de branchement en planification POCL procède comme suit :

- choisir une 'faille' dans un état non terminal σ.
- essayer de résoudre la faille choisie en appliquant les mécanismes de réparation disponibles jusqu'à atteindre un état terminal, c'est-à-dire un état incohérent (dans le cas où il contient une faille irréparable ou lorsque l'ensemble *ORD* relatif à cet état est incohérent) ou un état but (un état cohérent ne contenant aucune faille).

Les mécanismes de réparation sont liés aux types de failles détectées.

Les préconditions ouvertes

Pour une action α et un lien causal α'[p]α dans σ, ce type de faille est résolu en choisissant une action α' qui supporte p et en ajoutant ensuite α'[p]α à l'ensemble *CL* des liens causaux. On doit aussi ajouter l'action α' à *STEPS* si elle n'y appartient pas.

Les menaces

Une action $\alpha \in STEPS$ menace un lien causal $\alpha_1[p]\alpha_2$ de *CL* si elle peut retirer la précondition p de ce lien. Cette faille est résolue en ajoutant à l'ensemble *ORD* la contrainte « α précède α_1 ($\alpha \leq \alpha_1$) » ou la contrainte « α_2 précède α ($\alpha_2 \leq \alpha$) ».

Une action $\alpha_1 \in STEPS$ menace une autre action α_2 de *STEPS* si ces deux actions sont mutex. Cette faille est résolue en ajoutant à l'ensemble *ORD* la contrainte « α_1 précède α_2 ($\alpha_1 \leq \alpha_2$) » ou la contrainte « α_2 précède α_1 ($\alpha_2 \leq \alpha_1$) ».

b. Exemples de planificateurs POCL

- Le planificateur RePOP [Kambhampati et Nguyen 2001] est une variante du planificateur UCPOP [Penberthy et Weld 1992] utilisant une adaptation des heuristiques de calcul des distances et de la manipulation des contraintes disjonctives pour la planification d'ordre partiel.

- Le planificateur VHPOP [Younes et Simmons 2003] est un planificateur POCL temporel qui utilise une adaptation du procédé de branchement du planificateur UCPOP au cadre temporel. Il utilise en plus des fonctions heuristiques temporelles appliquées aux algorithmes de recherche Hill-Climbing, A* et IDA*.

c. Apports de CPT pour la planification POCL

Le planificateur CPT présente une extension de la planification POCL pour le cas temporel en ajoutant, pour chaque action $\alpha \in STEPS$, les variables T(α) et $dur(\alpha)$ représentant respectivement le temps initial et la durée de celle-ci. En fixant un majorant B pour la durée d'exécution du plan (qui peut être considérée comme un temps-limite), le domaine de T(α) est l'intervalle $[0, B\text{-}dur(\alpha)]$; en plus, on a $T(START)=0$, $T(END)=B$ et $dur(START)=dur(END)=0$.

Dans cette extension, un état du planificateur CPT est de la forme σ (*STEPS*, ORD_T, *CL*, *OPEN*,T(.)) où l'ensemble *ORD* qui représente l'ensemble des contraintes de précédence qualitatives est remplacé par les variables T(α) | α∈*STEPS* et l'ensemble des contraintes temporelles ORD_T.

Pour la résolution des failles, certains traitements sont ajoutés selon le type de faille :

- dans le cas des préconditions ouvertes pour une action α et un lien causal α'[p]α dans σ, après avoir choisi une action α' qui supporte p, la variable T(α') est créée et la contrainte temporelle suivante est ajoutée à ORD_T: T(α')+*dur*(α')≤T(α).

- dans le cas d'une action α qui menace un lien causal α_1[p]α_2, on doit ajouter dans l'ensemble ORD_T la contrainte « T(α)+*dur*(α)≤T(α_1) » ou la contrainte « T(α_2) + *dur*(α_2) ≤ T(α) ».

- dans le cas d'une action α_1 en mutex avec une autre action α_2, on doit ajouter dans l'ensemble ORD_T la contrainte « T(α_1)+*dur*(α_1)≤T(α_2) » ou la contrainte « T(α_2) + *dur*(α_2) ≤ T(α_1) ».

Classiquement, les planificateurs POCL basés sur des contraintes raisonnent uniquement sur les actions du plan courant. Ainsi, dans la majorité des cas, pour une action α, rien ne peut être inféré avant que cette action ne soit prise en considération pour être insérée dans le plan. Ceci constitue une faiblesse qui a conduit à des résultats peu satisfaisants. Pour cette raison, les développeurs du planificateur CPT ont cherché à déterminer des règles d'inférence sur les actions qui ne sont pas encore incluses dans le plan. Les notions suivantes ont été introduites :

- S(p,α) représente l'action qui supporte la précondition p de α.

- T(p,α) représente le temps initial (potentiellement indéterminé) de l'action relative à S(p,α).

- le lien causal α'[p]α devient une contrainte S(p,α) = α' et le support α' d'une précondition p de α commence alors à T(p,α) = T(α').

c. Formulation

La formulation de base du planificateur CPT pour la programmation par contraintes est composée de quatre parties : le prétraitement, les variables, les contraintes et le branchement.

Le prétraitement

Dans cette phase, le planificateur CPT calcule pour chaque action $\alpha \in O$ et chaque paire d'atomes {p,q} les heuristiques $H^2_T(\alpha)$ et $H^2_T(\{p,q\})$ qui sont les mêmes que celles du planificateur TP4. Plusieurs informations en sont déduites :

- des minorants sur le temps pour lequel les préconditions de l'action α sont satisfaites depuis l'état initial S_0.

- les paires d'atomes {p,q} pour lesquelles $H^2_T(\{p,q\})=\infty$; ces paires sont aussi appelées des mutex structurels.

L'heuristique H^1_T est utilisée pour déterminer les distances entre les actions selon le formalisme de Chen et Van Beek [Chen et Van Beek 1999] : pour chaque action $\alpha \in O$, l'heuristique $H^1_T(\alpha)$ est calculée depuis la situation initiale $S_0(\alpha)$ qui comprend tous les faits sauf ceux e-retirés par α. (voir définition 2.3) et pour toutes les actions $\alpha' \in O$, les distances $dist(\alpha,\alpha')$ sont alors initialisées aux valeurs résultantes $H^1_T(\alpha')$; elles constituent des minorants sur l'écart entre la fin de l'exécution de α et le début de celle de α' pour tous les plans valides dans lesquels α' suit α.

Les distances reliant les actions α à l'action *START* ($dist(START,\alpha)$) sont données par le calcul de $H^2_T(\alpha)$.

Les distances reliant les actions α à l'action *END* ($dist(\alpha,END)$) sont équivalentes au coût du plus court chemin qui relie *END* à α dans le graphe de pertinence (voir définition 2.2).

Définition 2.2

Un graphe de pertinence est un graphe pour lequel :

- les nœuds sont les actions de *O*, l'action *END* et le nœud initial.

- un arc $\alpha \rightarrow \alpha'$ signifie que α' produit une précondition p de α (α' est alors qualifiée de pertinente pour α).

Le coût $\delta(\alpha',\alpha)$ est alors donné par $\delta(\alpha',\alpha)= dur(\alpha')+dist(\alpha',\alpha)$.

Définition 2.3

Une action α e-retire un atome p quand elle retire p ou elle ajoute un atome q qui est en exclusion mutuelle avec p ou si l'une des préconditions de α est mutex avec p et α n'ajoute pas p.

Les variables

La formulation des contraintes traitées par le planificateur CPT nécessite la définition de plusieurs variables et la précision de leurs domaines respectifs :

- des variables sont définies pour chaque action $\alpha \in O$ et pour chaque précondition p de l'action α. Chacune de ces variables est définie dans un domaine comme suit : pour une variable X, le domaine est représenté par $\mathcal{D}[X]$ ou $X::[X_{min},X_{max}]$ (avec X_{min} et X_{max} sont respectivement des minorants et des majorants de X).

- pour une action $\alpha \in O$, le temps initial T(α) est représenté par une variable définie dans l'intervalle [0,+∞], en plus, nous avons T(*START*)=0.

- le support d'une précondition p de l'action α est représenté par la variable $S(p,\alpha)$ ayant pour domaine initial $\mathcal{D}[S(p,\alpha)]=\mathcal{A}(p)$ (avec $\mathcal{A}(p)$ l'ensemble des actions de O qui ajoutent p).

- le temps initial de $S(p,\alpha)$, noté $T(p,\alpha)$ est défini dans l'intervalle [0,+∞].

- la variable *InPlan*(α):: [0,1] relative à l'action α indique si celle-ci est présente ou pas dans le plan courant. On a *InPlan*(*START*)=InPlan(*END*)=1

- la variable *STEPS* représente l'ensemble des actions du plan courant ; autrement dit $STEPS = \{\alpha \in O \mid InPlan(\alpha)=1\}$.

Si l'action α n'est encore pas introduite dans le plan, les variables qui lui sont associées ainsi qu'à ses préconditions (c'est-à-dire T(α), $S(p,\alpha)$ et $T(p,\alpha)$ où p est une précondition de α) ne sont significatives que sous l'hypothèse qu'elles fassent partie du plan ; elles sont ainsi qualifiées de conditionnelles.

Les contraintes

Dans le planificateur CPT, les contraintes sont essentiellement temporelles, elles sont propagées par consistance des bornes ; par exemple si on a $T_{max}(\alpha)<T_{min}(\alpha')$, on peut déduire que l'action α précède l'action α' pour toutes les valeurs cohérentes choisies pour T(α) et T(α'). On note alors T(α)<T(α').

En plus, certaines contraintes sont appliquées à toutes les actions $\alpha \in O$ et pour toute précondition $p \in \mathcal{P}rec(\alpha)$:

- bornes : $\forall \alpha \in O$ on a :

$$\begin{cases} T(START)+dist(START,\alpha) \leq T(\alpha) \\ \text{et} \\ T(\alpha)+dist(\alpha,END) \leq T(END) \end{cases}$$

- préconditions: Le support α' d'une précondition $p \in \mathcal{P}rec(\alpha)$ doit précéder l'action α d'une quantité qui dépend de $\delta(\alpha,\alpha')$ (avec $\delta(\alpha,\alpha')=dur(\alpha)+dist(\alpha,\alpha')$) :

$$\begin{cases} T(\alpha) \geq Min_{\alpha' \in \mathcal{D}[S(p,\alpha)]} (T(\alpha') + \delta(\alpha',\alpha)) \\ \text{et} \\ T(\alpha') + \delta(\alpha',\alpha) > T(\alpha) \rightarrow S(p,\alpha) \neq \alpha' \end{cases}$$

- Liens causaux : $\forall \alpha \in O$, $p \in \mathcal{P}rec(\alpha)$ et α' qui e-retire p on a α' précède $S(p,\alpha)$ ou α précède α' :

$$\begin{cases} T(\alpha') + dur(\alpha') + Min_{\alpha'' \in \mathcal{D}[S(p,\alpha)]} dist(\alpha',\alpha'') \leq T(p,\alpha) \\ \text{ou} \\ T(\alpha) + \delta(\alpha,\alpha') \leq T(\alpha') \end{cases}$$

- mutex : $\forall (\alpha,\alpha') \in O^2$ | α et α' sont mutex, on a

$$\begin{cases} T(\alpha)+ \delta(\alpha,\alpha') \leq T(\alpha') \\ \text{ou} \\ T(\alpha) + \delta(\alpha',\alpha) \leq T(\alpha) \end{cases}$$

- support : $\forall \alpha \in O$ et $p \in \mathcal{P}rec(\alpha)$ on a :

$$\begin{cases} S(p,\alpha) = \alpha' \rightarrow T(p,\alpha) = \alpha' \\ \text{et} \\ T(p,\alpha) \neq T(\alpha') \rightarrow S(p,\alpha) \neq \alpha' \\ \text{et} \\ \text{Min}_{\alpha' \in \mathcal{D}[S(p,\alpha)]}\, T(\alpha') \leq T(p,\alpha) \leq \text{Max}_{\alpha' \in \mathcal{D}[S(p,\alpha)]}\, T(\alpha') \end{cases}$$

- la propagation des contraintes impliquant des variables liées à des actions incluses dans le plan courant (appelées variables *In-Plan*) sont propagées d'une manière classique pour laquelle si un domaine devient vide alors une incohérence est détectée.

- si une contrainte implique une variable liée à une action qui ne peut pas appartenir au plan courant (appelées variables *Out-Plan*), elle n'est pas propagée.

- les contraintes qui impliquent des variables conditionnelles (variables qui ne sont ni *In-Plan* ni *Out-Plan*) associées à la même action α et par conséquent soumises à la même hypothèse (que l'action α fasse partie du plan) sont propagées uniquement dans la direction des variables conditionnelles pour garder l'assurance que le domaine d'une variable conditionnelle ne dépend que de l'hypothèse selon laquelle cette variable est dans le plan. Par conséquent, si le domaine d'une variable conditionnelle associée à une action α devient vide, on peut inférer que α ne peut pas faire partie du plan courant (mais on ne peut pas inférer que ce plan est inconsistant) et dans ce cas, la valeur de la variable *InPlan*(α) est mise à 0 et pour chaque support $S(p,\alpha')$ telles que l'action α ajoute p celle-ci est enlevée de son domaine $\mathcal{D}[S(p,\alpha')]$. D'un autre côté, dans le cas où $S(p,\alpha') = \alpha$ est vrai pour une action α' incluse dans le plan, la valeur de la variable *InPlan*(α) est mise à 1.

Le branchement

Le branchement se fait en sélectionnant une faille dans un état σ ; une division binaire (notée $[C_1 ; C_2]$ où C_1 et C_2 sont des contraintes) est alors créée. Ensuite, le premier fils σ_1 de σ est obtenu en ajoutant C_1 à σ et en fermant le résultat par les règles de propagation. Le second fils σ_2 de σ est généré en ajoutant la contrainte C_2 quand la recherche avec σ_1 échoue. Les divisions binaires générées varient selon le type de la faille détectée :

- menace de support : une action α' menace un support S(p,α) quand les deux actions α et α' sont dans le plan courant, α' e-retire une précondition p de α et ni la contrainte « T_{min}(α') + dur(α') ≤ T_{min}(p,α) », ni la contrainte « T_{min}(α) + dur(α) ≤ T_{min}(α') » ne sont vraies. Dans ce cas, la division suivante est générée :

[T(α)+dur(α') + $Min_{\alpha'' \in \mathcal{D}[S(p,\alpha)]}$ $dist$(α',α'') ≤ T(p,α) ; T(α) + δ(α,α') ≤ T(α')].

- préconditions ouvertes : S(p,α) est une précondition ouverte quand la contrainte « |$\mathcal{D}$[S(p,α)]|>1 » est vraie pour une action α dans le plan courant. Dans ce cas, la division suivante est générée pour une action α' donnée :

[S(p,α) = α' ; S(p,α) ≠ α'].

- menaces de mutex : deux actions α et α' constituent une menace de mutex quand elles sont toutes les deux dans le plan courant, elle sont des effets qui interfèrent et ni la contrainte « T_{min}(α') + dur(α') ≤ T_{min}(α) », ni la contrainte « T_{min}(α) + dur(α) ≤ T_{min}(α') » ne sont vraies. Dans ce cas la division suivante est générée pour le couple (α,α') :

[T(α) + δ(α,α') ≤ T(α') ; T(α') + δ(α',α) ≤ T(α)].

Le schéma de branchement suivant est correct grâce à la validité du plan $\mathcal{P}$ obtenu depuis un état consistant σ sans aucune faille en ordonnançant les actions α_i à leurs temps initiaux minimaux respectifs t_i=T_{min}(α_i). Il est aussi complet grâce à la validité des règles de propagation et des disjonctions $C_1 \vee C_2$ associées à chaque division binaire [C_1 ; C_2].

Il existe plusieurs versions du planificateur CPT. Les premières ont été implémentées en utilisant le langage de programmation CLAIRE [Caseau et al. 1999 ; Caseau et Laburthe 1996] et la bibliothèque CHOCO [Laburthe 2000] pour la programmation par contraintes. Les nouvelles versions ont été programmées en utilisant le langage C.

Le planificateur CPT gère les actions canoniques, c'est-à-dire celles qui s'exécutent au maximum une seule fois durant le plan. En plus, dans les dernières versions, il est capable de gérer les actions non canoniques grâce à un mécanisme de clonage des actions faisant exécuter plusieurs instances (ou clones) d'une même action dans un seul plan.

Grâce à son mécanisme avancé de branchement et à ses règles pertinentes d'élagage, le planificateur CPT donne des résultats très intéressants dans les problèmes de planification

temporelle notamment par ses plans optimaux et parallèles et il a été récompensé plusieurs fois dans les compétitions internationales de planification depuis sa création. Cependant, il présente un inconvénient résidant en son incapacité de gérer aucune forme de ressources.

2. LES RESSOURCES DANS L'ORDONNANCEMENT ET DANS LA PLANIFICATION TEMPORELLE

Plusieurs approches de la planification temporelle et de l'ordonnancement visent à intégrer les ressources dans les problèmes liés au temps vu la ressemblance et l'interdépendance entre la notion de temps et celle de ressources.

Certains formalismes de planification et d'ordonnancement considèrent le temps et les ressources comme des variables numériques, d'autres essayent de traiter le temps et les ressources séparément en distinguant les caractéristiques spécifiques à chacune.

2.1. Représentation des ressources dans les problèmes d'ordonnancement

Les ressources sont présentes dans plusieurs problèmes d'ordonnancement, notamment dans les RCPSP [Brucker et al. 1999] (Resource Constrained Project Scheduling Problems ou Problèmes d'Ordonnancement de Projets sous Contraintes de Ressources). Elles sont généralement exprimées avec des contraintes numériques traitant des variables entières et elles sont utilisées conjointement avec les contraintes temporelles qui expriment les relations de précédence entre les tâches.

Dans ce contexte, la programmation par contraintes s'avère très prometteuse et a permis de réaliser un progrès considérable grâce aux nouveaux programmes spécialisés, notamment le système CHIP [Aggoun et al. 1998] qui est un outil de programmation par contraintes deuxième génération offrant un système de résolution performant basé sur le concept de Contraintes Globales. Il permet ainsi de traiter un grand nombre de problèmes d'ordonnancement avec ressources comme la planification du personnel, l'ordonnancement de production,…

2.1.1. Typologie des ressources

Dans l'ordonnancement, les ressources peuvent être réparties en trois catégories :

- **les ressources réservoir :** une ressource réservoir possède une disponibilité initiale et une disponibilité maximale (qui peut être finie ou pas). Une tâche α utilisant une ressource r peut incrémenter sa disponibilité (dans ce cas, la disponibilité de la ressource ne doit pas dépasser un niveau maximal lié à la capacité de son réservoir) ou la décrémenter (dans ce cas, la disponibilité de la ressource possède un niveau minimal qui doit toujours être disponible pour pouvoir exécuter la tâche). En plus, une ressource réservoir est partageable puisqu'elle peut être utilisée par plusieurs tâches simultanément tout en respectant les contraintes de disponibilité minimale et maximale. Ce type de ressources peut représenter le carburant d'une voiture par exemple.

- **les ressources discrètes :** la consommation d'une ressource discrète peut être considérée comme un emprunt : la tâche α qui consomme une certaine quantité q d'une ressource partageable r bloque cette quantité durant l'intervalle de temps de son exécution et elle la libère à la fin. Ce type de ressources peut représenter les ressources humaines par exemple.

- **les ressources unaires :** une ressource unaire est une ressource non cumulative, c'est-à-dire que si une tâche α consomme une ressource unaire r, cette ressource est entièrement bloquée durant l'intervalle de temps d'exécution de α et aucune autre tâche α' consommant r ne peut s'exécuter durant cet intervalle de temps.

2.1.2. Contraintes de ressources

Dans l'ordonnancement, la relation entre les tâches et les ressources est exprimée par les contraintes de ressources : pour une tâche α et une ressource r données, les contraintes de ressources déterminent la demande de la tâche α d'une certaine quantité de la ressource r pour pouvoir s'exécuter et comment l'exécution de α va affecter la disponibilité de la ressource r dans le système.

Selon [Laborie 2003], une contrainte de ressources est définie par un n-uplet (α, r, q, *TE*) tel que :

- α représente la tâche concernée.

- r représente la ressource concernée.

- q est un entier représentant la quantité de r consommée (si $q<0$) ou produite (si $q>0$) par la tâche α.

- la variable *TE* (Time Extent) explique la façon dont la ressource r est consommée durant le temps d'exécution de la tâche α. Elle peut prendre une des six valeurs suivantes :

- *FromStartToEnd* : une contrainte de ressource de ce type indique que l'effet de la tâche α sur la disponibilité de la ressource r durera tout le temps de son exécution et ensuite, cette disponibilité redeviendra comme avant. Par exemple, pour une consommation ($q<0$), la disponibilité de r doit être au moins égale à $|q|$ dès l'exécution de α, et durant tout le temps d'exécution de cette tâche, la quantité $|q|$ de la ressource r sera bloquée.
- *AfterStart* : une contrainte de ressource de ce type indique que l'influence de la tâche α sur la disponibilité de la ressource r prendra effet dès que α commence son exécution et elle durera jusqu'à la fin de l'ordonnancement.
- *AfterEnd* : une contrainte de ressource de ce type indique que l'influence de la tâche α sur la disponibilité de la ressource r prendra effet dès que α finit son exécution et elle durera jusqu'à la fin de l'ordonnancement.
- *BeforeStart* : une contrainte de ressource de ce type indique que l'influence de la tâche α sur la disponibilité de la ressource r prend effet dès le début de l'ordonnancement et persiste jusqu'à ce que cette tâche commence son exécution. Par exemple, pour $q<0$, cette contrainte exprime le fait que la valeur $|q|$ est requise pour la disponibilité de la ressource r pour pouvoir exécuter la tâche α.
- *BeforeEnd* : une contrainte de ressource de ce type indique que l'influence de la tâche α sur la disponibilité de la ressource r prend effet dès le début de l'ordonnancement et continue jusqu'à ce que cette tâche termine son exécution.
- *Always* : une contrainte de ressource de ce type indique que l'influence de la tâche α sur la disponibilité de la ressource r prend effet durant tout le temps de l'ordonnancement du début jusqu'à la fin.

Les ressources peuvent avoir des propriétés communes dans les problèmes d'ordonnancement de même type. Par exemple, pour les RCPSP, les ressources ont les propriétés suivantes :

- toutes les ressources sont discrètes et cumulatives (elles peuvent être utilisées par plusieurs tâches simultanément). En plus, elles sont disponibles à tout instant en quantité limitée.

- l'utilisation d'une ressource par une tâche α est considérée comme un emprunt : la quantité utilisée par α est restituée à la fin de son exécution, la variable *TE* a donc une valeur commune *FromStartToEnd* pour toutes les ressources du problème.

2.2. Les ressources dans la planification temporelle

2.2.1. Approche PDDL

Comme nous avons vu précédemment, le langage PDDL est capable de décrire les problèmes de planification temporelle depuis sa version 2.1 [Fox et Long 2001]. Cette version permet aussi de décrire les contraintes numériques utilisées essentiellement pour représenter les ressources.

Le langage PDDL2.1 est décomposé en 4 niveaux d'expressivité, et c'est dans le deuxième niveau que les variables et les contraintes numériques sont introduites avec la possibilité de tester et de changer instantanément leur valeur. Les ressources peuvent donc être exprimées comme des contraintes numériques et les actions peuvent affecter les ressources de plusieurs manières :

- emprunt : une action peut emprunter une quantité d'une ressource au moment où elle commence son exécution, et à la fin de cette exécution elle doit restituer (totalement ou partiellement) la quantité empruntée.

- consommation : une action peut ne pas restituer la quantité de la ressource qu'elle a demandée pour son exécution à la fin de celle-ci. Ce phénomène est appelé consommation.

- incrémentation : une action peut générer une ressource par incrémentation. Dans ce cas, la disponibilité de la ressource concernée sera incrémentée d'une certaine quantité.

- assignation : une action peut assigner la disponibilité d'une ressource à un niveau donné, c'est-à-dire qu'à la fin de l'exécution de cette action, la quantité de ressource disponible est fixée à un nombre précis. Ce phénomène peut être observé par exemple dans l'action de remplissage à plein d'un réservoir : quel que soit le niveau de la ressource dans le réservoir avant l'exécution de l'action, il deviendra égal à la capacité maximale du réservoir à la fin.

En plus, le quatrième niveau d'expressivité de PDDL 2.1 traite le cas d'une interférence entre la manipulation des ressources et le temps, c'est-à-dire que le temps d'exécution de

l'action dépend de la quantité consommée (ou produite) d'une ressource donnée par cette action.

Enfin, il faut noter que PDDL 2.1 permet de définir les critères de préférence (optimalité) d'un plan valide, qui peut être le temps d'exécution du plan ou la quantité d'une ressource donnée (par exemple le carburant) ou même une fonction faisant intervenir le temps et les ressources.

Exemple : Le problème ZenoTravel

Le problème ZenoTravel peut être modélisé en intégrant certaines variables numériques, par exemple le nombre de passagers situés simultanément dans un avion, la quantité de carburant dans le réservoir d'un avion, la capacité maximale du réservoir de chaque avion, la distance entre les villes, la consommation moyenne d'un avion lors de son vol…

Le problème peut alors être étendu en prenant en compte le carburant et le nombre de passagers dans un avion :

- le vol d'une ville à une autre consomme une quantité de carburant qui dépend de la distance parcourue et de la vitesse du vol. En plus, la durée du vol elle-même dépend de cette distance et de la vitesse de l'avion.

- il y a la possibilité de recharger l'avion en carburant par l'action *refuel*. Dans ce cas, la capacité du réservoir de cet avion atteindra son maximal (recharge à plein). La durée de cette action dépend de la vitesse de recharge du carburant ainsi que de la quantité à recharger jusqu'à atteindre la capacité maximale.

- le nombre de passagers dans un avion est incrémenté par l'action *board* et il est décrémenté par l'action *debark*.

- il y a deux possibilités d'aller d'une ville à une autre : soit par un vol à vitesse normale (action *Fly*), soit par un vol à grande vitesse (action *Zoom*) qui consomme plus de carburant.

- il y a possibilité de spécifier les critères et les ressources à optimiser lors de la recherche d'un plan optimal.

Le codage du problème ZenoTravel avec le langage PDDL2.1 en intégrant les variables numériques indiquées est présenté à l'annexe C. Comme pour les problèmes PDDL, les données communes à toutes les instances d'un problème PDDL2.1 sont modélisées d'une façon paramétrée dans un fichier spécifique (voir annexe C-1) et les données spécifiant

chaque instance du problème (l'état initial, le but, les données numériques, les conditions d'optimalité…) sont modélisées dans un fichier particulier. L'annexe C-2 contient une modélisation d'une instance du problème avec un avion, deux personnes et trois villes avec une condition d'optimalité intégrant le temps et les ressources.

2.3. Représentation des ressources dans la planification temporelle

Dans la planification temporelle, la représentation et la manipulation des ressources diffèrent d'un planificateur à l'autre selon les capacités de celui-ci et selon son niveau d'expression.

2.3.1. Les ressources dans le planificateur IxTeT

Le planificateur IxTeT [Ghallab et Laborie 1995] est capable de gérer les contraintes temporelles et les contraintes de ressources consommables ; l'utilisation des ressources peut être :

- un emprunt d'une certaine quantité sur un intervalle temporel.
- une consommation d'une certaine quantité à un instant précis.
- une production d'une certaine quantité à un instant précis.

IxTeT gère les ressources non partageables qui ont une capacité de partage unaire et ne peuvent donc être utilisées que par une seule action à la fois et les ressources partageables banalisées qui ont une capacité de partage multiple dont on ne gère pas explicitement l'allocation car ce sont soit des ressources agrégées, soit des ressources de type continu.

Les connaissances dans IxTeT sont organisées autour d'une table à deux dimensions dont une représente le temps, l'autre les attributs et les ressources du domaine.

2.3.2. Les ressources dans le planificateur TP4

Le planificateur TP4 [Geffner et Haslum 2001] gère deux types de ressources : les ressources renouvelables et les ressources consommables.

Les ressources renouvelables : l'utilisation de telles ressources par une action est similaire à un emprunt d'une certaine quantité qui sera réservée à l'action durant son temps d'exécution, ensuite la quantité empruntée de la ressource est restaurée en sa totalité.

Les ressources consommables : contrairement aux ressources renouvelables, les ressources consommables ne sont pas restaurées à la fin de l'exécution de l'action et dans ce cas, il peut y avoir dans le problème des actions génératrices de la ressource consommable (si celle-ci n'est pas une ressource consommable d'une façon monotone).

Dans TP4, les ressources sont représentées avec des quantités numériques et le problème de planification avec ressources est étendu avec les ensembles $\mathcal{R}_p$ et C_p représentant respectivement les ressources renouvelables et consommables du problème.

Pour chaque ressource $r \in \mathcal{R}_p \cup C_p$, la quantité initialement disponible est représentée par $avail(r)$ et pour chaque action α, $use(r,\alpha)$ représente la quantité de r empruntée (si $r \in \mathcal{R}_p$) ou consommée (si $r \in C_p$) par α.

CONCLUSION

Dans la planification temporelle indépendante du domaine, même si les formalismes existants offrent un niveau d'expressivité assez satisfaisant pour la représentation du temps et des ressources, les planificateurs temporels développés ne permettent pas encore d'exploiter tout le pouvoir expressif de ces formalismes puisqu'ils présentent toujours des difficultés à garantir un bon compromis entre la qualité des plans trouvés (du point de vue optimalité, parallélisme, gestion de ressources limitées,…) et le temps de recherche.

Le but de cette thèse est d'élaborer un formalisme de gestion des ressources pour la planification temporelle qui permet de prendre en charge les ressources consommables et celles renouvelables par incrémentation ou par assignation. Il s'inspirera de la représentation des ressources existante dans les problèmes d'ordonnancement ainsi que les planificateurs temporels qui gèrent les ressources.

CHAPITRE 3

FORMALISME PROPOSE POUR LA REPRESENTATION DES RESSOURCES DANS LE PLANIFICATEUR CPT

INTRODUCTION

La gestion des ressources dans les planificateurs temporels actuels présente encore des faiblesses qui se manifestent particulièrement dans la qualité des plans trouvés. En effet, la plupart des planificateurs temporels qui gèrent les ressources imposent de fortes restrictions sur les ressources prises en considération. Certains planificateurs temporels traitent les problèmes de planification temporelle sans même prendre compte les ressources. On peut donner ici l'exemple du planificateur CPT [Geffner et Vidal 2004], qui, malgré ses performances exceptionnelles dans la résolution des problèmes temporels sans ressources, présente l'inconvénient majeur de ne pouvoir traiter aucune forme de ressources, ce qui influe d'une façon négative sur la qualité des plans qu'il arrive à trouver même si ceux-ci sont optimaux (par rapport au temps) et parallèles, car il risquent de ne pas être valides si on tient compte des ressources limitées.

Pour cette raison, nous avons visé à élaborer un formalisme permettant de représenter les ressources pour les planificateurs temporels en tenant compte de la consommation, l'emprunt, la production et l'assignation des ressources par les actions.

Ce formalisme sera exploité par le planificateur CPT afin d'acquérir la capacité de gérer les ressources dans toutes leurs formes : les ressources renouvelables, les ressources consommables, les ressources consommables d'une manière monotone,…

1. DESCRIPTION DU FORMALISME PROPOSE

Dans notre choix du formalisme de représentation des ressources, nous avons cherché à garder un compromis entre le niveau d'expressivité et l'efficacité ; nous avons observé la gestion des ressources dans la planification temporelle et dans l'ordonnancement et nous nous sommes rendus compte que les formalismes choisis dans les planificateurs temporels sont inspirés de l'ordonnancement ; notamment en ce qui concerne la typologie des ressources et les méthodes d'utilisation de la ressource par les actions.

Dans notre étude sur les formalismes utilisés dans la planification pour représenter les ressources, nous avons constaté que le formalisme élaboré dans PDDL 2.1 [Fox et Long, 2001] est très expressif car il permet, entre autres, de représenter les relations entre la

consommation (ou la production) d'une ressource par une action et le temps d'exécution de celle-ci. En plus, nous avons remarqué que dans les problèmes d'ordonnancement avec temps et ressources, les variables numériques représentant le temps et les ressources sont des variables entières et leurs domaines sont discrets. En revanche, dans la planification temporelle, ces variables sont rationnelles voire réelles dans certains planificateurs.

La gestion des ressources dans l'ordonnancement a bénéficié de ces simplifications pour obtenir des résultats très intéressants combinant le temps et les ressources. Cependant, dans la planification temporelle, les résultats sont encore assez limités, en particulier pour les problèmes combinant le temps et les ressources. En plus, il faut noter que les planificateurs qui exploitent toute l'expressivité de PDDL 2.1 sont très rares et que les résultats qu'ils fournissent sont loin d'être satisfaisants.

Pour ces raisons, nous avons choisi la représentation des ressources dans un formalisme inspiré des problèmes d'ordonnancement (en particulier les RCPSP [Brucker et al. 1999]) mais qui est plus riche, pour permettre de traiter certains cas particuliers codés en PDDL 2.1. Il est en relation avec le formalisme de Smith et Weld [Smith et Weld 1999] utilisé dans le planificateur CPT pour exprimer le temps. Il faut mentionner, cependant, que ce formalisme est moins expressif que PDDL 2.1 notamment en ce qui concerne l'interdépendance entre le temps et les ressources exprimée dans le quatrième niveau.

Dans ce formalisme, les ressources doivent respecter les règles suivantes.

Règle 3.1 : propriétés des ressources

Dans le cas général, les ressources du système sont cumulatives (elles peuvent être consommées simultanément par plusieurs actions) et disponibles, à tout instant, en quantité limitée. On note $Q_{bef}(r, \alpha)$ la disponibilité de la ressource r dans le système juste avant l'exécution d'une action α, $Q_{aft}(r, \alpha)$ la disponibilité de cette ressource dans le système juste après l'exécution de α et $Q_{max}(r)$ est la capacité maximale de la ressource r (capacité maximale du réservoir).

Règle 3.2 : manipulation des ressources par les actions

Pour être exécutée, une action α peut nécessiter un niveau minimum de la ressource r dans le réservoir, on le note $q_{inf}(r,\alpha)$, elle peut également nécessiter un niveau maximum $q_{sup}(r,\alpha)$ de cette ressource ou un niveau exact $q_{eq}(r,\alpha)$.

L'exécution d'une action α peut influer sur la disponibilité de chaque ressource $r \in \mathcal{R}$ (ensemble de toutes les ressources considérées dans un problème de planification) de trois manières :

- soit en incrémentant sa valeur de $q_{inc}(r, \alpha)$, c'est-à-dire qu'après l'exécution de α, la disponibilité de r devient égale à $Q_{bef}(r, \alpha) + q_{inc}(r, \alpha)$.
- soit en décrémentant (consommant) sa valeur de $q_{dec}(r, \alpha)$ c'est-à-dire qu'après l'exécution de α, la disponibilité de r devient égale à $Q_{bef}(r, \alpha) - q_{dec}(r, \alpha)$.
- soit en assignant sa valeur à une constante fixe $q_{as}(r, \alpha)$, c'est-à-dire qu'après l'exécution de α, la disponibilité de r devient égale à $q_{as}(r, \alpha)$.

Règle 3.3 : menaces relatives aux ressources

Une action α qui consomme une ressource cumulative $r \in \mathcal{R}$ peut menacer une autre action β consommant la ressource r si on a : $Q_{bef}(r, \beta) - q_{dec}(r, \alpha) < q_{inf}(r, \beta)$.

Toute action qui consomme une ressource unaire r' menace les autres actions consommatrices de cette ressource.

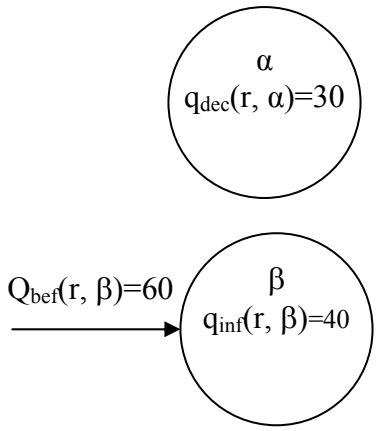

Figure 3.1: Menace entre deux actions : cas de consommation de ressource

Dans l'exemple présenté à la figure 3.1, on peut constater que l'action α, consommatrice de la ressource r menace l'action β puisque $Q_{bef}(r, \beta) - q_{dec}(r, \alpha) = 60-30=30$ est bien inférieur à $q_{inf}(r,\beta)=40$.

Une action α qui génère une ressource r par incrémentation peut menacer une autre action β si on a : $Q_{bef}(r, \beta) + q_{inc}(r, \alpha) > q_{sup}(r, \beta)$.

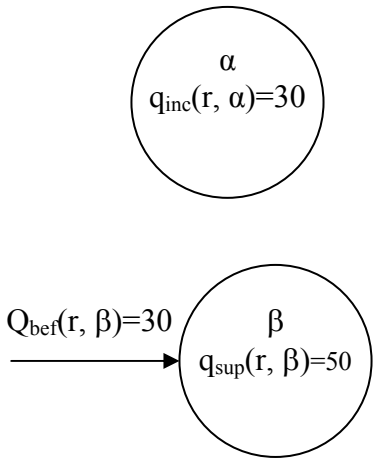

Figure 3.2: Menace entre deux actions : cas de production de ressource par incrémentation

Dans l'exemple présenté à la figure 3.2, on peut constater que l'action α, productrice de la ressource r par incrémentation menace l'action β puisque $Q_{bef}(r, \beta) + q_{inc}(r, \alpha) = 30+30=60$ est bien supérieur à $q_{sup}(r,\beta)=50$.

On peut résoudre le problème de menace relative à une ressource cumulative $r \in \mathcal{R}$ entre deux actions α et β qui consomment la ressource r ou la génèrent par incrémentation soit en cherchant une action permettant de modifier la disponibilité de la ressource r d'une manière suffisante pour éviter la cause de la menace soit en les mettant en exclusion mutuelle (mutex) interdisant ainsi tout parallélisme entre ces deux actions (voir définition 3.1).

Pour une ressource unaire r', le seul moyen de résoudre un problème de menace entre deux actions consommant cette ressource est de les mettre en exclusion mutuelle.

Une action α qui assigne la disponibilité d'une ressource cumulative $r \in \mathcal{R}$ à une valeur fixe $q_{as}(r,\alpha)$ peut menacer une autre action β si on a : $q_{as}(r, \alpha) \notin [q_{inf}(r, \beta), q_{sup}(r, \beta)]$. Le seul moyen pour résoudre le problème de menace dans ce cas est mettre les deux actions α et β en mutex.

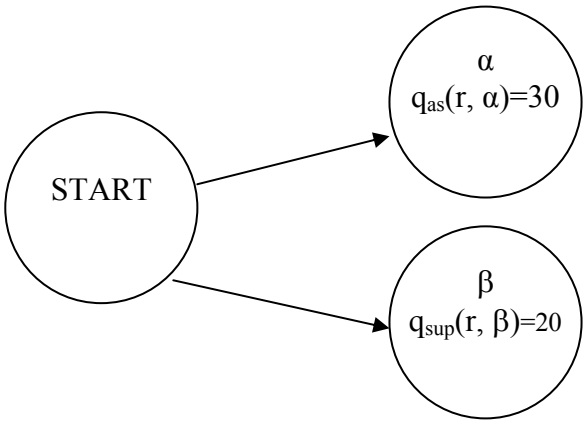

Figure 3.3: Menace entre deux actions : cas n°1 d'assignation de ressource

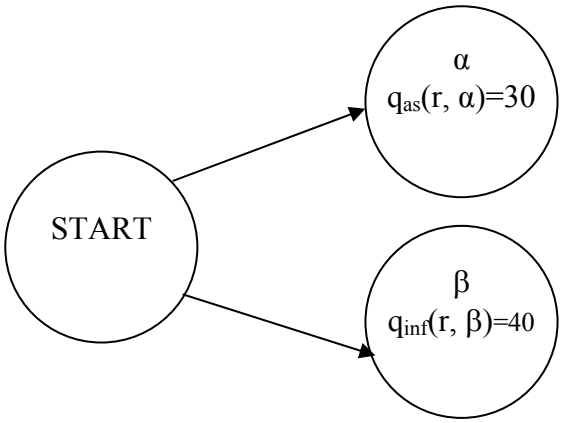

Figure 3.4: Menace entre deux actions : cas n°2 d'assignation de ressource

Dans ces deux exemples, les deux actions α et β se menacent mutuellement. Puisque l'action α est une action d'assignation, le seul moyen de résoudre le problème de menace est de mettre ces deux actions en exclusion mutuelle.

Règle 3.4 : les ressources et les actions START et END

Il existe deux actions fictives *START* et *END* de durées nulles représentant respectivement le début et la fin du plan. La disponibilité initiale des ressources dans le système est représentée comme résultat d'assignation par l'action *START*. Ainsi, si par exemple la disponibilité initiale du système en une ressource r est q_0, on a $q_0 = q_{as}(r, START)$. Il est utile de noter qu'il n'y a aucun niveau minimal ou maximal de disponibilité requis pour une ressource r par l'action *START*.

Règle 3.5 : plan valide – plan optimal

Un plan valide est un plan respectant les différentes contraintes du problème de planification, notamment les contraintes de précédence pour tout couple d'actions $(\alpha, \beta) \in \mathcal{A}^2$ (ensemble des actions possibles dans un problème de planification) et résolvant à tout instant t, les éventuels conflits dans l'utilisation de chaque ressource r (la somme des quantités de ressources requises pour chaque action α correspond à la disponibilité de la ressource juste avant l'exécution de α).

Le critère d'optimisation recherché dans notre travail étant le temps, il est exigé que tout plan valide fasse évoluer le système de l'état initial à l'état final en un minimum de temps.

Définition 3.1

On dit que deux actions α_1 et α_2 sont reliées entre elles avec une relation de parallélisme et on note $\alpha_1 \parallel \alpha_2$ si et seulement si α_1 et α_2 s'exécutent simultanément pendant au moins une unité de temps.

Remarque 3.1

Si deux actions α_1 et α_2 sont en exclusion mutuelle (on dit aussi qu'elles sont en mutex), elles ne peuvent pas être reliées par une relation de parallélisme. Dans ce cas, il faut ordonner leur exécution selon les autres relations de précédence pour confirmer que α_1 précède α_2 ($\alpha_1 < \alpha_2$) ou α_2 précède α_1 ($\alpha_2 < \alpha_1$).

2. TYPOLOGIE DES RESSOURCES

2.1. Cas de consommation

Dans notre formalisme, la différence entre les ressources renouvelables et les ressources consommables gérées par les logiciels d'ordonnancement et par la plupart des planificateurs gérant les ressources peut être exprimée par les variables Q_{aft} , Q_{bef} et q_{inf} :

- pour une ressource totalement renouvelable r on a pour toute action α qui la consomme : $Q_{aft}(r, \alpha)=Q_{bef}(r, \alpha)$.

- pour une ressource partiellement renouvelable r on a pour toute action α qui la consomme : $q_{inf}(r, \alpha) <Q_{aft}(r, \alpha)<Q_{bef}(r, \alpha)$.

- pour une ressource consommable r on a pour toute action α qui la consomme : $q_{inf}(r,\alpha)=Q_{aft}(r, \alpha)$.

Il existe cependant un cas particulier de ressources consommables utilisé notamment dans le planificateur TP4 [Geffner et Haslum 2001], c'est le cas des ressources consommables d'une manière monotone pour lesquelles il n'existe aucune action qui puisse les régénérer dans le problème de planification. La distinction de ce cas permet dans certains exemples d'améliorer le temps de recherche des plans valides. En effet, en traitant les ressources consommables de façon monotone à part dans les branchements, on peut réduire d'une manière significative le nombre de retours-arrière car dans chaque niveau, on peut éliminer du plan partiel toutes les actions qui demandent une quantité de ressources supérieure à leur disponibilité dans ce niveau puisque ces actions ne pourront être introduites dans aucun plan valide.

2.2. Cas de production

Dans notre travail, nous nous intéressons particulièrement à deux cas de production des ressources : la production par incrémentation et la production par assignation. La différence entre ces deux types se situe dans la manière avec laquelle une action de production d'une ressource agit sur elle :

- une action α qui produit une ressource r par incrémentation agit sur celle-ci en incrémentant sa disponibilité de $q_{inc}(r, \alpha)$.

- une action α qui produit une ressource r par assignation agit sur celle-ci en assignant sa disponibilité à une valeur fixe $q_{as}(r, \alpha)$ indépendante de la disponibilité avant l'exécution de α.

Remarque 3.2

Une même ressource peut être générée par incrémentation par certaines actions et par assignation par d'autres actions appartenant au même problème de planification.

Cette distinction dans la manière de produire les ressources est très importante car cette donnée, comme nous allons voir plus loin, a une influence capitale sur les mécanismes de propagation et de branchement suivis.

Pour une recharge par assignation, on connaît à l'avance le résultat qui est indépendant de la quantité de ressource dans le réservoir avant l'exécution de l'action de recharge.

Pour une recharge par incrémentation, le résultat dépend de la disponibilité de la ressource concernée.

3. CONTRAINTES DE RESSOURCES

Il est important d'exprimer les règles considérées dans notre formalisme de représentation des ressources sous forme de contraintes pour chercher à les satisfaire lors de la phase de recherche d'un plan valide.

Les contraintes prises en considération dans notre formalisme sont les suivantes :

(C3.1) Les ressources sont cumulatives ou unaires et elles sont disponibles, à tout instant de l'exécution du plan, en quantité limitée.

(C3.2) Pour chaque ressource $r \in \mathcal{R}$ on doit avoir une capacité maximale $Q_{max}(r)$ qui peut être finie ou non. Cela signifie que pour toute action α du plan on a toujours : $Q_{bef}(r,\alpha) \leq Q_{max}(r)$ et $Q_{aft}(r,\alpha) \leq Q_{max}(r)$

(C3.3) Pour les ressources cumulatives, on peut exécuter parallèlement plusieurs actions consommant la même ressource, mais il faut que la somme des demandes de la ressource r pour les actions exécutées en parallèle à chaque instant t ne dépasse pas la disponibilité de la ressource à cet instant.

On peut exprimer cette contrainte comme suit :

toutes les actions qui s'exécutent entre deux actions α_1 et α_2 qui génèrent une même ressource r (avec $\alpha_1 < \alpha_2$) ne peuvent pas demander une quantité de la ressource r qui dépasse la quantité $Q_{aft}(r, \alpha_1)$, c'est-à-dire :

$$\sum_{(ai \in At) et (ai > \alpha 1) et (ai < \alpha 2)} q_{inf}(r,a_i) \leq Q_{aft}(r, \alpha_1)$$

(où $\mathcal{A}_t$ est l'ensemble des activités a_i en cours d'exécution à l'instant t).

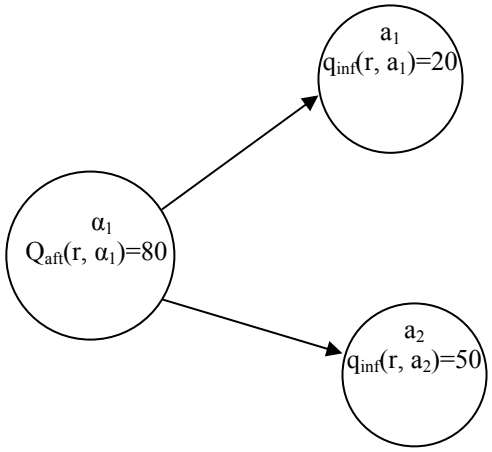

Figure 3.5: Parallélisme entre actions consommatrices de la même ressource cumulative

Dans l'exemple donné à la figure 3.5, les actions a_1 et a_2 peuvent s'exécuter en parallèle car leur demande cumulée de la ressource r $q_{inf}(r,a_1)+ q_{inf}(r,a_2)=20+50=70$ ne dépasse pas la disponibilité de cette ressource juste après l'exécution de l'action α_1 ($Q_{aft}(r,\alpha_1)=80$) qui les précède. On ne peut pas cependant exécuter une autre action a_3 demandant 20 unités de r ($q_{inf}(r,a_3)=20$) en parallèle avec les actions a_1 et a_2 car on aurait dans ce cas $q_{inf}(r,a_1)+ q_{inf}(r,a_2)+ q_{inf}(r,a_3)=90> Q_{aft}(r,\alpha_1)$.

(C3.4) Les actions étant non interruptibles, la quantité $q_{inf}(r,\alpha)$ de chaque ressource r nécessaire pour l'exécution de l'action α est bloquée pour cette action tout au long de l'exécution de celle-ci.

D'autres contraintes relatives aux effets des actions de consommation et de recharge de ressources seront détaillées dans les prochains chapitres car elles sont dépendantes de la solution choisie pour résoudre le problème de planification avec ressources.

CONCLUSION

Dans ce chapitre, nous avons présenté le formalisme que nous avons élaboré pour intégrer les ressources dans les planificateurs temporels optimaux. Il profite des avantages constatés dans la représentation des ressources dans les problèmes d'ordonnancement tout en gardant un niveau d'expressivité acceptable pour les problèmes de planification temporelle.

Ce formalisme est moins expressif que celui présenté dans PDDL 2.1 notamment en ce qui concerne le quatrième niveau d'expressivité mais il est compatible avec le formalisme de Smith et Weld pour la représentation des données temporelles, utilisé par un grand nombre de planificateurs temporels.

Nous allons maintenant nous efforcer d'appliquer le formalisme élaboré au planificateur CPT tout en conservant sa puissance dans la résolution des problèmes de planification temporelle. Pour cela, nous allons établir des méthodes de gestion des ressources basées sur le mécanisme suivi par le planificateur CPT, c'est-à-dire la détection et la résolution des éventuels conflits relatifs à l'utilisation (production ou consommation) de la même ressource par plusieurs actions simultanées.

Il faut en plus noter que dans le cadre de ce travail, le temps reste le seul critère d'optimisation des plans trouvés. C'est-à-dire que le meilleur plan est celui dont la durée totale d'exécution est la plus courte.

CHAPITRE 4

GESTION DES RESSOURCES DANS CPT : METHODE BASEE SUR LES LIENS DE RESSOURCES

INTRODUCTION

Nous avons introduit, dans le chapitre précédent, le formalisme que nous avons établi pour prendre les ressources en considération. Dans ce chapitre, nous allons proposer une première méthodologie pour la gestion des ressources dans le planificateur CPT en respectant les règles et contraintes fixées par le formalisme.

Les ressources consommables d'une manière monotone nécessitent un traitement spécifique qui est indépendant de la solution choisie pour traiter les autres types de ressources. Ce traitement sera expliqué en premier lieu.

Pour les autres types de ressources, la méthodologie proposée, dans ce chapitre, est basée sur un nouveau concept : les liens de ressources qui permettent au planificateur de détecter les éventuelles menaces entre actions consommant ou produisant la même ressource.

1. PRISE EN CHARGE DES RESSOURCES CONSOMMABLES D'UNE MANIERE MONOTONE

1.1. Règle d'élagage

Dans le formalisme que nous avons choisi, la seule façon d'augmenter la disponibilité d'une ressource est d'exécuter des actions qui la génèrent par assignation ou par incrémentation. Or, pour le cas d'une ressource r consommable d'une manière monotone, il n'existe pas d'actions qui puissent la générer. C'est pourquoi il n'est pas possible d'augmenter la disponibilité de cette ressource pendant l'exécution d'un plan.

Une action ne peut être exécutée que si les ressources qu'elle nécessite sont disponibles en quantité suffisante. Cependant, dans le cas d'une ressource r consommable d'une manière monotone, la disponibilité ne peut que diminuer au cours de l'exécution d'un plan, c'est pourquoi on peut exclure définitivement une action α du plan solution si elle nécessite une quantité $q_{inf}(r,\alpha)$ de la ressource r qui dépasse la quantité $Q_{bef}(r,\alpha)$ de cette ressource disponible dans le système avant son exécution.

Cette règle d'élagage permet de réduire significativement le nombre de retours-arrière et réduire ainsi le temps de recherche dans un grand nombre de cas.

1.2. Implémentation et résultats expérimentaux

Puisque le planificateur CPT ne gère pas les ressources dans sa version initiale, la première modification à effectuer est d'introduire une structure pour représenter toutes les ressources du problème de planification. Cette structure doit contenir la liste des actions qui consomment cette ressource et la liste de celles qui la génèrent (pour les actions consommables d'une manière monotone, cette liste ne contient que l'action *START*).

En plus, la structure représentant les actions dans le programme CPT doit être enrichie pour pouvoir gérer les ressources : on doit y ajouter la liste des ressources consommées par cette action avec, pour chaque ressource, les quantités q_{inf} et q_{dec} qui lui sont relatives pour l'action considérée. On doit aussi ajouter dans cette structure la liste des ressources générées par cette action avec, pour chaque ressource, les quantités q_{sup} et q_{inc} (ou q_{as}) qui lui sont relatives pour l'action considérée. Pour l'action *START*, cette deuxième liste contient les ressources initialement disponibles avec leurs quantités initiales respectives.

Puisque le remplissage de toutes ces structures se fait au niveau du parseur qui permet d'analyser le problème de planification codé en PDDL 2.1. Il est nécessaire d'effectuer des modifications sur ce programme pour qu'il puisse analyser les données relatives aux ressources.

Le mécanisme de gestion des règles d'élagage dans le planificateur CPT permet d'implémenter ce test sans difficulté. Il suffit juste de détecter les ressources pour lesquelles il n'existe aucune action génératrice et de comparer la disponibilité de chacune avec la quantité demandée par toute action qui la consomme avant l'exécution de celle-ci.

Pour des actions α_1, α_2,…, α_n s'exécutant en parallèle[2] et consommant la même ressource r, la somme des quantités de r qu'elles nécessitent ne doit pas dépasser la disponibilité de la

[2] Dans la planification temporelle et l'ordonnancement, deux actions s'exécutent en parallèle si l'intersection de leurs intervalles de temps d'exécution respectifs n'est ni vide ni ponctuelle.

ressource r avant l'exécution de chacune d'elles. Ce qui peut être exprimé par la contrainte suivante: $\forall j \in \{1,\ldots,n\}$ on a : $\sum_{i=1}^{n} q_{inf}(r,\alpha_i) \leq Q_{aft}(r, \alpha_j)$.

Cette solution a été testée sur des instances du problème ZenoTravel ne possédant pas d'actions *Refuel* et avec un certain nombre de ressources :

Nombre d'avions	Nombre de villes	Nombre de ressources consommables de manière monotone	Temps de recherche moyen dans CPT initial (en secondes)	Temps de recherche moyen dans CPT modifié (en secondes)
1	3	1	0.15	0.12
2	3	2	0.16	0.16
2	4	2	0.16	0.17
3	5	3	4.1	3.24
3	6	3	27.43	21.13

Tableau 4.1: Expérimentation du planificateur CPT prenant en compte les ressources consommables d'une manière monotone

Nous avons choisi comme ressource consommable d'une manière monotone le carburant relatif à un avion. Pour cette raison, nous avons une équivalence, dans toutes les instances du problème, entre le nombre de ressources consommables et le nombre d'avions.

D'après ces résultats, on peut remarquer que dans certains cas, le temps de recherche s'est légèrement amélioré pour les problèmes ayant un plan valide respectant les contraintes de ressources introduites, en particulier lorsque le plan trouvé par le planificateur CPT original coïncide avec celui trouvé par le planificateur CPT modifié. La raison de cette amélioration est que l'espace de recherche peut se réduire à chaque étape par l'élimination des actions dès que les quantités de ressources qu'elles demandent dépassent leurs disponibilités respectives.

Pour cette même raison, il y a un gain de temps de recherche dans les cas où le problème n'admet aucune solution valide qui respecte les contraintes temporelles.

Dans les cas où le planificateur CPT original trouve un plan valide (et optimal au niveau temporel) plus rapidement que le planificateur CPT modifié, le plan trouvé par le planificateur original est réfuté par la nouvelle version du planificateur car il ne respecte pas les contraintes de ressources introduites dans les modifications. On peut donc dire que dans ce cas, même s'il

y a une certaine perte de temps dans la recherche de plans valides et optimaux, il y a un gain dans la qualité des plans trouvés (s'il en existe un qui respecte toutes les contraintes de ressources introduites). Il est enfin possible d'aboutir à un échec conduisant à la conclusion qu'il n'existe aucun plan qui respecte toutes les contraintes du problème de planification (notamment les contraintes de ressources)

2. PRISE EN CHARGE DES RESSOURCES CONSOMMABLES ET RENOUVELABLES

2.1. Restrictions admises

La méthode que nous proposons dans ce chapitre pour donner au planificateur CPT la capacité de gérer les ressources consommables et renouvelables repose sur la prise en compte des contraintes suivantes :

(C4.1) Les actions étant non interruptibles, la quantité $q_{inf}(r,\alpha)$ de chaque ressource r nécessaire pour l'exécution d'une action α est bloquée pour celle-ci tout au long de son exécution.

(C4.2) Si une action α génère une ressource r (par incrémentation ou par assignation), cette ressource est bloquée pour l'action α durant son exécution, c'est-à-dire qu'aucune action β consommatrice ou génératrice de la ressource r ne peut s'exécuter pendant l'exécution de l'action α.

(C4.3) A la fin de l'exécution d'une action α consommatrice d'une ressource r, la disponibilité $Q_{aft}(r, \alpha)$ de cette ressource se décrémente de la somme des consommations de toutes les autres actions qui consomment la même ressource r et qui ont été exécutées au cours de l'exécution de α (puisque aucune action génératrice de r ne peut être exécutée dans cet intervalle de temps), c'est-à-dire : $Q_{aft}(r, \alpha) = Q_{bef}(r, \alpha) - \sum_{(ai \in A) \& (ai < \alpha)} q_{dec}(r, a_i)$.

(C4.4) A la fin de l'exécution d'une action α génératrice d'une ressource r par incrémentation, la disponibilité $Q_{aft}(r,\alpha)$ de la ressource r augmente de $q_{inc}(r,\alpha)$, c'est-à-dire : $Q_{aft}(r,\alpha) = Q_{bef}(r, \alpha) + q_{inc}(r,\alpha)$.

(C4.5) A la fin de l'exécution d'une action α génératrice d'une ressource r par assignation, la disponibilité $Q_{aft}(r,\alpha)$ de la ressource r devient égale à $q_{as}(r,\alpha)$, c'est-à-dire $Q_{aft}(r,\alpha) = q_{as}(r,\alpha)$.

2.2. Introduction des liens de ressources dans CPT

Comme mentionné dans le chapitre 2, une action $\alpha \in STEPS$ menace un lien causal $\alpha_1[p]\alpha_2$ de *CL* si elle peut retirer la précondition p de ce lien. Cette faille est résolue en ajoutant à l'ensemble *ORD* la contrainte « α précède α_1 $(\alpha \leq \alpha_1)$ » ou la contrainte « α_2 précède α $(\alpha_2 \leq \alpha)$ ». Ceci ne peut pas être appliqué aux ressources à cause du caractère cumulatif de celles-ci. En effet, plusieurs actions consommant la même ressource r peuvent être exécutées en parallèle si la disponibilité de la ressource r le permet. Dans le cas échéant, on ne parle plus de préconditions retirées car la menace est sur le point de vue numérique.

En plus, une action α qui consomme une ressource r peut menacer un ensemble d'actions $\{\alpha_1, \alpha_2,\ldots, \alpha_n\}$ qui consomment la ressource r et qui peuvent s'exécuter en parallèle même si cette action α peut s'exécuter en parallèle avec chaque action $\alpha_i \in \{\alpha_1, \alpha_2,\ldots, \alpha_n\}$. Par exemple, si la disponibilité de la ressource r est de 50 unités et si on a : $q_{inf}(r, \alpha_1)= q_{inf}(r, \alpha_2)= q_{inf}(r, \alpha_3)=\ldots= q_{inf}(r, \alpha_9)=5$, on peut conclure que les actions $\{\alpha_1, \alpha_2,\ldots, \alpha_9\}$ peuvent s'exécuter en parallèle si elles ne sont pas mutex car leur consommation totale de la ressource r est égale à $45<50$. Si, en plus, on a une action α qui consomme 10 unités de la ressource r, alors cette action menace l'ensemble $\{\alpha_1, \alpha_2,\ldots, \alpha_9\}$ car la consommation totale devient égale à $55>50$. Par contre, la disponibilité de la ressource r permet d'exécuter l'action α en parallèle avec chaque action $\alpha_i \in \{\alpha_1, \alpha_2,\ldots, \alpha_9\}$.

Tout ceci montre l'inefficacité des liens causaux à détecter et à traiter les menaces liées aux ressources, ce qui explique la nécessité d'introduire une nouvelle structure au planificateur CPT.

La structure introduite est nommée les liens de ressources ; ces liens sont similaires aux liens causaux gérés par le planificateur CPT avec certaines différences dues au caractère cumulatif des ressources.

2.2.1. Notion de lien de ressources

Un lien de ressources est défini comme suit : étant données n actions α_1, α_2, … ,α_n et une ressource r telles que α_2, …, α_n sont des actions génératrices de r (par assignation ou par incrémentation) et α_1 est consommatrice de r, on définit le lien de ressources R(r, α_1) par son domaine $\mathcal{D}$[R(r, α_1)] = {α_2, …, α_n,}. Ceci explique que l'une des actions α_2, …, α_n doit nécessairement s'exécuter avant α_1 pour générer les ressources demandées en précondition :

$$\forall\ \alpha \in \mathcal{A}, \forall\ r \in \mathcal{R} \text{ tq } q_{inf}(r, \alpha)>0 \text{ on a : } \mathcal{D}[R(r, \alpha)] = \{a \in \mathcal{A} / (q_{inc}(r,a)>0) \text{ v } (q_{as}(r,a)>0)\}$$

2.2.2. Menace de lien de ressources

La principale différence entre un lien causal et un lien de ressources réside dans la détection des actions qui les menacent et dans la façon de gérer les éventuelles menaces.

Avant d'introduire les menaces des liens de ressources, il est bon de se souvenir de la contrainte (C4.2) indiquant que si une action α génère une ressource r, cette ressource est bloquée pour l'action α durant son exécution.

Pour pouvoir gérer les ressources dans le planificateur CPT, on doit détecter et traiter les éventuelles menaces entre actions consommant ou produisant la même ressource. Suivant le type de l'action qui génère la ressource, on peut distinguer deux cas de menaces:

a. Cas d'une recharge par assignation :

Etant donné un ensemble d'actions {α_1,…, α_n}$\subseteq \mathcal{A}$ consommant la même ressource r$\in \mathcal{R}$ et une action α' qui assigne la disponibilité de r à $q_{as}(r,\alpha')$, un lien de ressources R(r,α_i)=α' est menacé par une action α si et seulement si :

$$\begin{cases} \{\alpha_1, \ldots, \alpha_n, \alpha'\} \subseteq \{a \,/\, \mathit{InPlan}(a)\} \\ \text{et} \\ \sum_{a \in \{\alpha 1, \ldots, \alpha n, \alpha\}} q_{inf}(r, a) > q_{as}(r, \alpha') \end{cases}$$

Dans ce cas, α' est exclu du domaine du lien R(r, α) et l'action α doit être placée après une autre action $\beta \in \mathcal{D}[R(r, \alpha)]$.

On rappelle que la variable *InPlan*(α) relative à une action α indique si celle-ci est présente ou pas dans le plan courant.

Remarque 4.1

Pour une ressource r donnée, on peut exclure du domaine du lien de ressources relatif à chaque action α qui nécessite une quantité $q_{inf}(r, \alpha)$ de r les actions de recharge par assignation qui assignent la disponibilité de r à une quantité inférieure à $q_{inf}(r, \alpha)$, ce qui constitue une règle d'élagage utile dans beaucoup de cas.

Illustration

On suppose avoir quatre actions faisant partie du plan partiel α_1, α_2, α_3, α_4 qui consomment une ressource r et deux actions α' et α" qui génèrent cette ressource par assignation.

Les données numériques sont les suivantes : $q_{inf}(r, \alpha_1)=30$, $q_{inf}(r, \alpha_2)=40$, $q_{inf}(r, \alpha_3)=20$, $q_{inf}(r, \alpha_4)=30$, $q_{as}(r, \alpha')=100$ et $q_{as}(r, \alpha'')=50$.

Chaque action α_i qui consomme la ressource r a un lien de ressources dont le domaine est donné comme suit: $\mathcal{D}[R(r, \alpha_1)] = \mathcal{D}[R(r, \alpha_2)] = \mathcal{D}[R(r, \alpha_3)] = \mathcal{D}[R(r, \alpha_4)] = \{\alpha', \alpha''\}$

Dans le mécanisme de branchement du planificateur CPT, on suppose que les actions sont sélectionnées pour faire partie du plan selon l'ordre suivant α_1, α_2, α_3 et enfin α_4 (en réalité, cet ordre dépend des heuristiques implémentées dans le planificateur). En ajoutant l'action α_1, le lien $R(r, \alpha_1)$ exige l'ajout d'une action du domaine $\mathcal{D}[R(r, \alpha_1)]$ pour la précéder. Cette action est sélectionnée par les mécanismes de branchement de CPT. On suppose dans

l'exemple que l'action α' est choisie. On peut schématiser ce raisonnement dans un graphe orienté où les nœuds représentent les actions et les arcs représentent les liens de ressources :

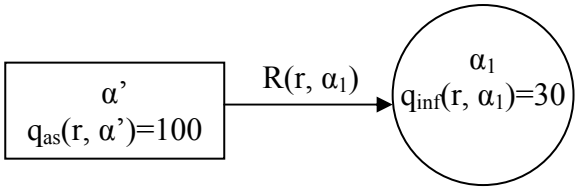

Figure 4.1: Schématisation de la précédence par lien de ressources pour la recharge par assignation - étape n°1

L'insertion des actions α_2 et α_3 ne pose pas de problème car ces deux actions ne menacent pas $R(r, \alpha_1)$ puisqu'on a $q_{inf}(r, \alpha_1) + q_{inf}(r, \alpha_2) + q_{inf}(r, \alpha_3) = 90 < q_{as}(r, \alpha')$. Puisque l'action α' fait déjà partie du plan partiel, elle sera choisie pour précéder les actions α_2 et α_3, et on aura dans le graphe de planification :

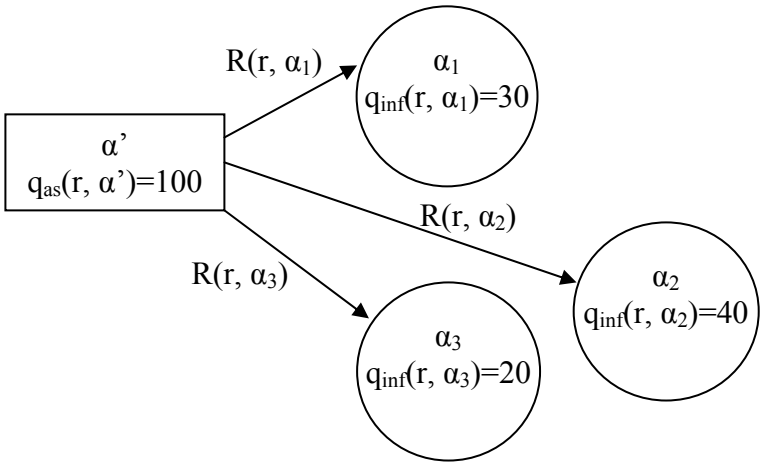

Figure 4.2: Schématisation de la précédence par lien de ressources pour la recharge par assignation - étape n°2

L'insertion de l'action α_4 menace les liens de ressources $R(r, \alpha_1)$, $R(r, \alpha_2)$ et $R(r, \alpha_3)$ car on a $q_{inf}(r, \alpha_1) + q_{inf}(r, \alpha_2) + q_{inf}(r, \alpha_3) + q_{inf}(r, \alpha_3) = 120 > q_{as}(r, \alpha')$. Dans ce cas, l'action α' sera éliminée du domaine du lien de ressources $R(r, \alpha_4)$ qui devient alors réduit au singleton {α"}. L'action α" est alors insérée dans le plan pour précéder l'action α_4 et, selon le mécanisme de branchement du planificateur CPT et les autres contraintes liées au problème de planification, deux cas peuvent se présenter :

- l'action α_4 est placée pour précéder l'action α' ce qui sera schématisé comme suit (la flèche en trait interrompu représente un lien de précédence):

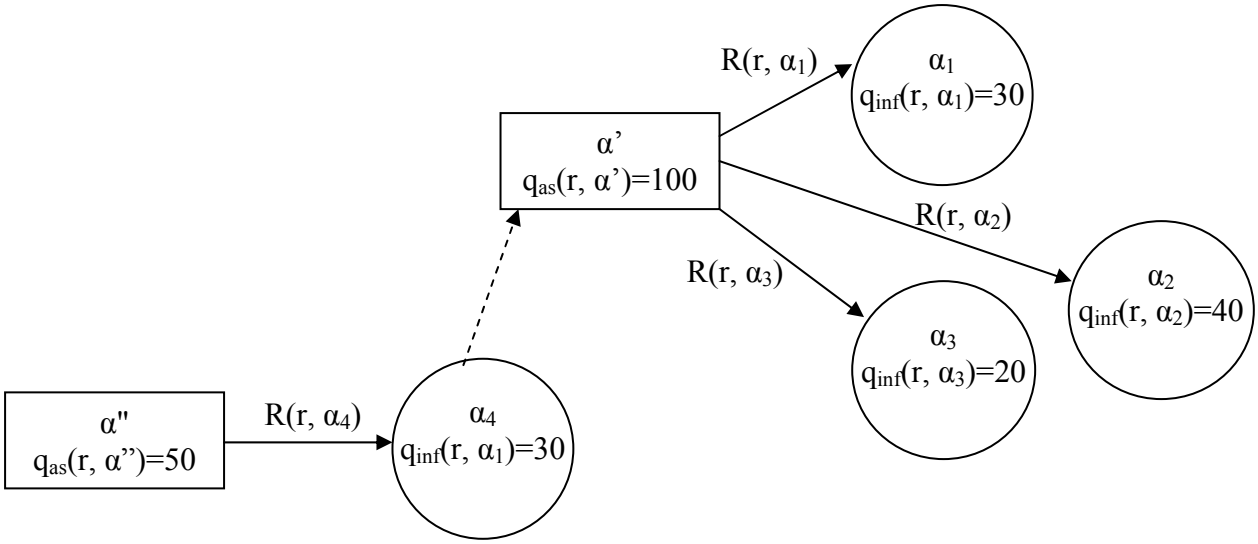

Figure 4.3: Schématisation de la précédence par lien de ressources pour la recharge par assignation - étape n°3

- l'action α" est placée pour suivre les actions α_1, α_2 et α_3 ce qui sera schématisé comme suit :

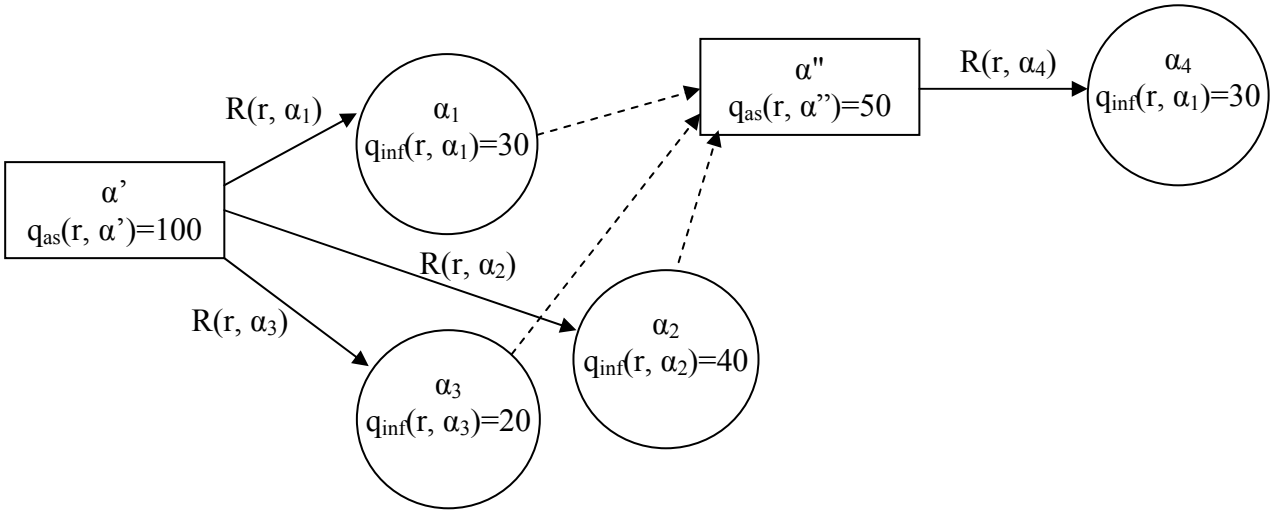

Figure 4.4: Schématisation de la précédence par lien de ressources pour la recharge par assignation - étape n°4

Application au problème ZenoTravel

Dans l'exemple de ZenoTravel, il y a plusieurs types de ressources (les passagers, le carburant,…). On prend en considération ici le carburant qui est consommé par les actions *Fly* et *Zoom* et rechargé par assignation grâce à l'action *Refuel*.

Les données numériques sont les suivantes : il y a un seul avion dans le problème, l'action *Refuel* assigne le réservoir de cet avion à 50 unités de carburant, le réservoir contient initialement 25 unités de carburant et les actions *Fly* et *Zoom* nécessitent respectivement 15 et 20 unités de carburant.

Le schéma suivant montre comment on doit procéder pour pouvoir exécuter deux actions *Fly* et deux actions *Zoom* par le même avion:

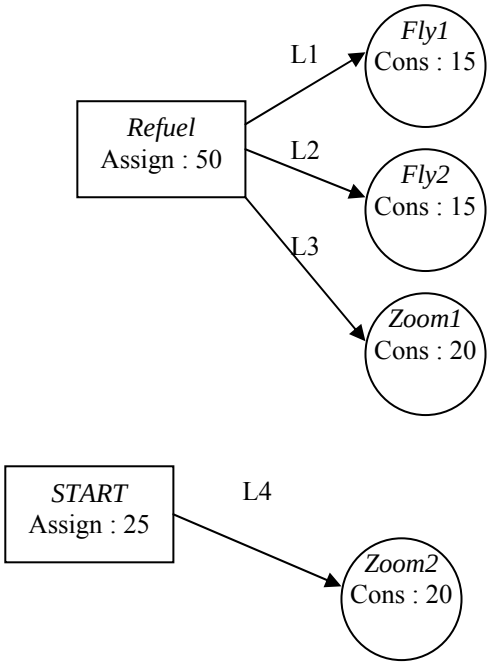

Figure 4.5: Illustration de la méthode utilisant les liens de ressources pour le problème ZenoTravel - Cas d'une recharge par assignation

L'action *Refuel* génère une quantité suffisante de carburant pour exécuter les actions *Fly1*, *Fly2* et *Zoom1* (liens de ressources L1, L2 et L3), mais l'action *Zoom2* constitue une menace pour ces liens de ressources car elle ne trouve pas assez de carburant pour être exécutée ; on doit donc supprimer l'action *Refuel* du domaine du lien L4. Ce domaine se réduit donc à l'action *START*.

b. Cas d'une recharge par incrémentation :

Etant donné un ensemble d'actions $\{\alpha_1,\ldots, \alpha_n\}\subseteq \mathcal{A}$ consommant la même ressource $r\in\mathcal{R}$ et une action α' qui incrémente la disponibilité de r à la quantité $q_{inc}(r, \alpha')$, un lien de ressources $R(r, \alpha_i)= \alpha'$ est menacé par une action α si et seulement si :

$$\begin{cases} \{\alpha_1,\ldots, \alpha_n, \alpha\} \subseteq \{a \,/\, \mathit{InPlan}(a)\} \\ \text{et} \\ \sum_{a\in\{\alpha 1,\ldots,\alpha n,\alpha\}} q_{inf}(r, a) > Q_{bef}(r, \alpha') + q_{inc}(r, \alpha') \end{cases}$$

Dans ce cas, α' n'est pas obligatoirement exclue du domaine du lien R(r, α). En effet, s'il existe une autre action α'' génératrice de la ressource r telle que :

$$\begin{cases} \sum_{a\in\{\alpha 1,\ldots,\alpha n,\alpha\}} q_{inf}(r, a) \le Q_{bef}(r, \alpha') + q_{inc}(r, \alpha') + q_{inc}(r, \alpha'') \\ \quad \text{(si } \alpha' \text{ précède } \alpha'') \\ \text{ou} \\ \sum_{a\in\{\alpha 1,\ldots,\alpha n,\alpha\}} q_{inf}(r, a) \le Q_{bef}(r, \alpha'') + q_{inc}(r, \alpha'') + q_{inc}(r, \alpha') \\ \quad \text{(si } \alpha'' \text{ précède } \alpha') \end{cases}$$

alors l'action α' et l'action α'' font toutes les deux partie du plan partiel.

Pour résoudre les menaces sur les liens de ressources en cas de recharge par incrémentation, une solution est d'effectuer les changements suivants lorsqu'un lien de ressources $R(r, \alpha_i)= \alpha'$ est menacé par une action α:

- on affecte la valeur 0 à $q_{inc}(r, \alpha')$, à $Q_{bef}(r, \alpha')$ et à $q_{inf}(r, a)$ $\forall$ $a\in\{\alpha_1,\ldots, \alpha_n, \alpha\}$,
- l'action α' devient alors consommatrice de la ressource r et on a donc :

$$q_{inf}(r, \alpha') = \Sigma \sum_{a\in\{\alpha 1,\ldots,\alpha n,\alpha\}} q_{inf}(r, a) - Q_{bef}(r, \alpha') + q_{inc}(r, \alpha').$$

En d'autres termes, l'action α' demande la quantité de ressources qui lui manque pour pouvoir exécuter les actions $\alpha_1,\ldots, \alpha_n$, ainsi que l'action α.

On s'arrête lorsqu'on atteint l'un des deux cas suivants :

- la menace est relevée et dans ce cas toutes les actions de recharge essayées font partie du plan, nous obtenons à la fin : $Q_{aft}(\alpha) = \sum_{a \in \{\alpha', \alpha'', \ldots, \alpha^{(n)}, \alpha\}} Q_{bef}(r, a)$

$+ \sum q_{inc}(r, a / a \in \{\alpha', \alpha'', \ldots, \alpha^{(n)}\}) - \sum_{a \in \{\alpha', \alpha'', \ldots, \alpha^{(n)}, \alpha\}} q_{inf}(r,a)$

ou

- le domaine du lien de ressources $\mathcal{D}[R(r, \alpha')]$ devient vide sans que la menace ne soit relevée. Dans ce cas, on effectue un retour-arrière pour essayer d'autres branchements.

Illustration

On suppose avoir deux actions faisant partie du plan partiel α_1, α_2 qui consomment une ressource r et trois actions α', α'' et $\alpha^{(3)}$ qui génèrent cette ressource par incrémentation.

Les données numériques sont les suivantes : $q_{inf}(r, \alpha_1)=50$, $q_{inf}(r, \alpha_2)=15$, $q_{inc}(r, \alpha')=30$, $q_{inc}(r, \alpha'')=40$, $q_{inc}(r, \alpha^{(3)})=30$.

Chaque action α_i qui consomme la ressource r a un lien de ressources dont le domaine est donné comme suit: $\mathcal{D}[R(r, \alpha_1)] = \mathcal{D}[R(r, \alpha_2)] = \{\alpha', \alpha'', \alpha^{(3)}\}$.

Dans le mécanisme de branchement du planificateur CPT, on suppose que l'action α_1 est sélectionnée pour faire partie du plan avant α_2. En ajoutant l'action α_1, le lien $R(r, \alpha_1)$ exige l'ajout d'une action du domaine $\mathcal{D}[R(r, \alpha_1)]$ pour la précéder. On peut remarquer qu'aucune des trois actions du domaine $\mathcal{D}[R(r, \alpha_1)]$ n'est capable de générer la quantité de la ressource r nécessaire pour exécuter α_1, cependant aucune de ces actions n'est encore exclue du domaine de $\mathcal{D}[R(r, \alpha_1)]$. On suppose dans l'exemple que l'action α' est choisie. La schématisation de ce raisonnement dans un graphe donne:

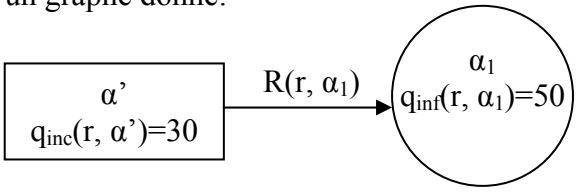

Figure 4.6: Schématisation de la précédence par lien de ressources pour la recharge par incrémentation - étape n°1

Puisque l'action α' ne génère par assez de ressources pour exécuter α_1 on doit procéder comme suit :

- la valeur de $Q_{bef}(r, \alpha_1)$ est incrémentée de $q_{inc}(r, \alpha')$.
- on affecte la valeur 0 à $q_{inc}(r, \alpha')$, $Q_{bef}(r, \alpha')$ et $q_{inf}(r, \alpha_1)$.
- l'action α' devient consommatrice de la ressource r et on a donc : $q_{inf}(r, \alpha') = 50 - 30 = 20$.

Puisque l'action α' est devenue consommatrice de r, elle doit avoir un lien de ressource correspondant dont le domaine est : $\mathcal{D}[R(r, \alpha')] = \{\alpha'', \alpha^{(3)}\}$ (par souci de clarté dans l'illustration, on suppose que toutes les actions sont canoniques, c'est-à-dire qu'une même action ne peut être exécutée qu'une seule fois dans tout le plan). On suppose que l'action α" est alors choisie pour précéder l'action α'. Le graphe de planification est alors :

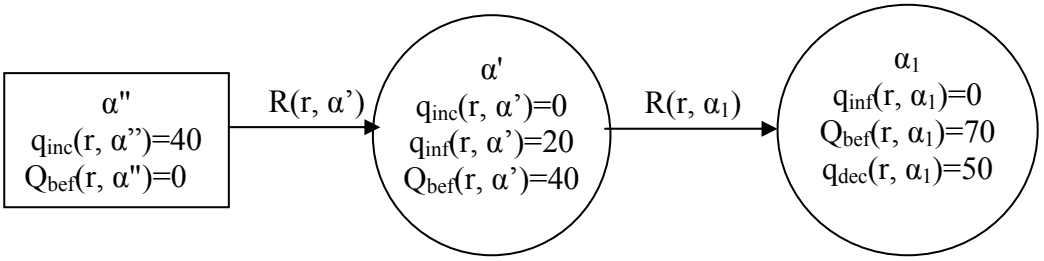

Figure 4.7: Schématisation de la précédence par lien de ressources pour la recharge par incrémentation - étape n°2

Grâce à ce processus, on a pu générer une quantité de la ressource r suffisante pour exécuter l'action α_1. En plus, la disponibilité du réservoir après l'exécution de l'action α_1 devient égale à 20. Cette quantité peut être ensuite utilisée pour exécuter une autre action consommatrice de la ressource r. Dans notre exemple, l'action α_2 nécessite 15 unités de la ressource r pour être exécutée, ce qui donne la possibilité de la mettre dans le plan après l'action α_1. Le graphe final du processus est :

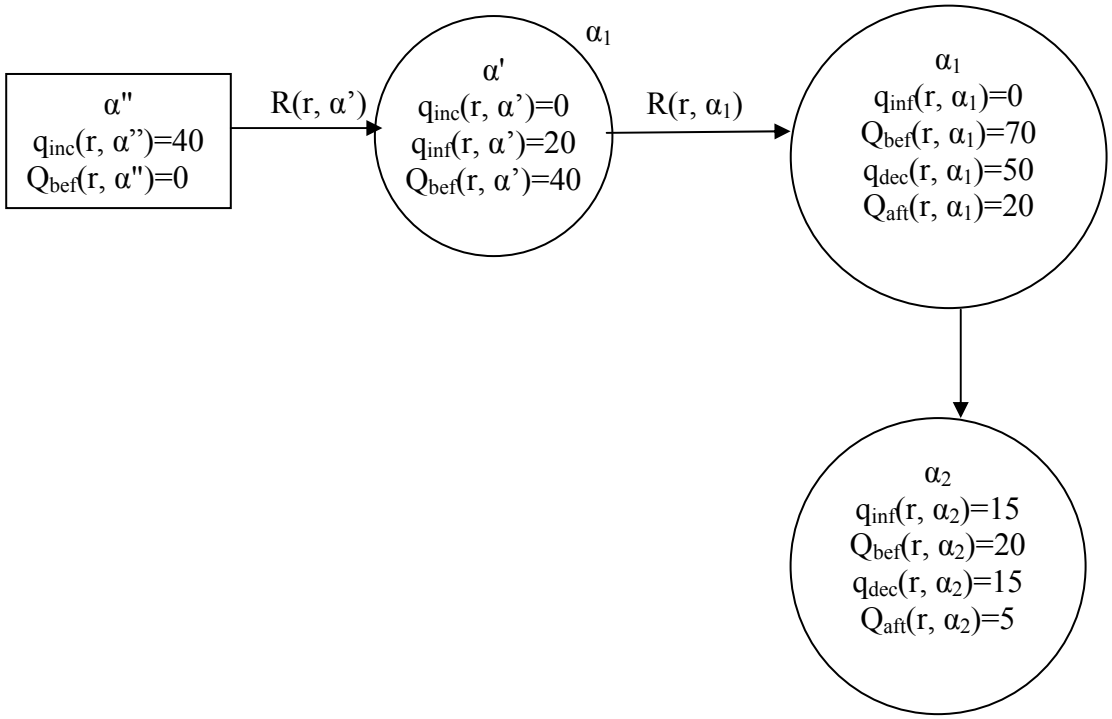

Figure 4.8: Schématisation de la précédence par lien de ressources pour la recharge par incrémentation - étape n°3

Application au problème ZenoTravel

On considère dans l'exemple de ZenoTravel qu'il y a un avion et deux actions de recharge *Refuel* et *Wrefuel* générant respectivement 10 et 20 unités de carburant pour le seul avion du problème et l'action Fly nécessitant 25 unités de carburant. Le schéma suivant montre comment on doit procéder pour pouvoir exécuter l'action *Fly* (dans cet exemple restreint, nous ne nous intéressons pas à l'embarquement des passagers dans l'avion):

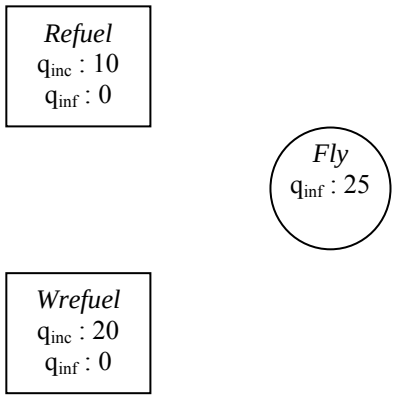

Figure 4.9: Illustration de la méthode utilisant les liens de ressources pour le problème ZenoTravel - Cas d'une recharge par incrémentation – étape n°1

On constate aisément qu'aucune des deux actions *Refuel* et *Wrefuel* ne peut produire toute seule une quantité suffisante de carburant pour exécuter l'action *Fly*. On effectue alors la transformation indiquée ci-dessous :

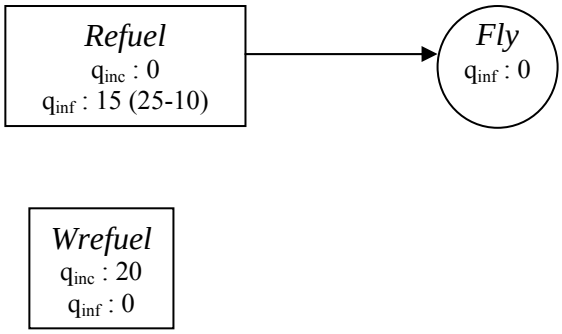

Figure 4.10: Illustration de la méthode utilisant les liens de ressources pour le problème ZenoTravel - Cas d'une recharge par incrémentation – étape n°2

L'action *Refuel* demande désormais 5 unités de r, qui lui seront fournies par l'action *Wrefuel*, nous obtenons à la fin le plan partiel suivant :

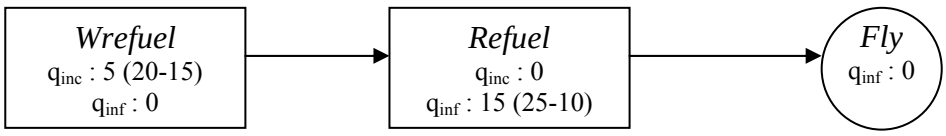

Figure 4.11: Illustration de la méthode utilisant les liens de ressources pour le problème ZenoTravel - Cas d'une recharge par incrémentation – étape n°3

On arrive enfin à satisfaire la demande en ressources de l'action *Fly*, en plus on a maintenant 5 unités dans le réservoir qui pourraient servir pour d'autres actions qui consomment du carburant.

2.3. Implémentation dans le planificateur CPT

Pour pouvoir intégrer la notion de ressources renouvelables dans le planificateur CPT, on doit établir, pour chaque action α du problème, la liste des ressources consommées et celle des ressources générées par α. On doit aussi indiquer, pour chaque ressource r du problème, la

liste des actions qui la génèrent et celle des actions qui la consomment. Ces listes sont utiles pour faire le lien entre actions et ressources.

On doit en plus établir, pour chaque action α, les liens de ressources correspondant à chaque ressource consommée par α.

On peut distinguer un lien de ressources par rapport à un lien causal en se focalisant sur le type de fluents qui constituent le lien. En effet, pour un lien de ressources, le fluent indique la production de la ressource en question. Par exemple, pour une ressource r le fluent indiquant son rechargement peut être appelé « *r_produced* ». Ce fluent doit être demandé par toutes les actions qui consomment la ressource r.

Après avoir distingué les liens de ressources des liens causaux, on doit détecter les menaces engendrées par les actions sur chaque lien de ressources.

La résolution de ces menaces s'effectue automatiquement par les mécanismes de filtrage et de retour-arrière présents dans le planificateur CPT.

Dans le cas d'incrémentation, les actions qui génèrent la ressource r demandent elles-mêmes le fluent *r_produced* mais avec une quantité nulle. Lorsqu'on a une menace de lien de ressource $R(r, \alpha_i)= \alpha'$, la quantité de r demandée par l'action α' passe de 0 à $\sum_{a\in\{\alpha 1,\ldots,\alpha n,\alpha\}} q_{inf}(r,a) - Q_{bef}(r, \alpha') + q_{inc}(r, \alpha')$ (les actions $\alpha_1,\ldots, \alpha_n$ sont les actions qui consomment la ressource r et une action parmi elles est celle qui constitue la menace du lien de ressources) et le planificateur procède comme indiqué dans le paragraphe précédent.

2.4. Résultats expérimentaux

2.4.1. Premier exemple : le problème ZenoTravel

Nous avons testé le planificateur obtenu sur plusieurs instances du problème ZenoTravel dans sa version numérique trouvées dans les benchmarks de la compétition IPC3 car elles représentent une bonne illustration de l'intégration des ressources rechargeables par assignation et par incrémentation à différents niveaux de complexité.

Les deux tableaux suivants présentent les résultats trouvés pour le cas de ressources rechargeables respectivement par assignation et par incrémentation.

Nombre d'avions	Nombre de villes	Nombre de ressources rechargeables par assignation	Temps de recherche moyen dans CPT modifié (en secondes)
1	3	1	0.21
2	3	2	0.24
2	4	2	0.41
3	5	3	5.33
3	6	3	37.17

Tableau 4.2: Expérimentation du planificateur CPT prenant en compte les ressources rechargeables par assignation dans le problème ZenoTravel

Nombre d'avions	Nombre de villes	Nombre de ressources rechargeables par incrémentation	Temps de recherche moyen dans CPT modifié (en secondes)
1	3	1	0.33
2	3	2	0.41
2	4	2	0.78
3	5	3	7.24
3	6	3	46.25

Tableau 4.3: Expérimentation du planificateur CPT prenant en compte les ressources rechargeables par incrémentation dans le problème ZenoTravel

Le tableau 4.4 présente les résultats expérimentaux de l'effet des modifications effectuées sur le planificateur CPT pour le problème de planification ZenoTravel sans ressources. Nous avons mis en comparaison le temps moyen de recherche écoulé par la version initiale de CPT et le temps moyen de recherche écoulé par la version modifiée du même planificateur.

Nombre d'avions	Nombre de villes	Temps de recherche moyen dans CPT initial (en secondes)	Temps de recherche moyen dans CPT modifié (en secondes)
1	3	0.15	0.21
2	3	0.16	0.24
2	4	0.16	0.41
3	5	4.1	5.33
3	6	27.43	37.17

Tableau 4.4: Comparaison entre la version originelle et la version modifiée du planificateur CPT pour le traitement des problèmes de planification temporelle sans ressources dans le problème ZenoTravel

Nous avons choisi 5 minutes comme valeur de temps limite, ce qui ne suffit pas à résoudre la majorité des instances impliquant quatre avions et plus même dans la version initiale de

CPT. C'est pour cette raison que les résultats indiqués dans les tableaux sont limités aux instances impliquant trois avions au maximum.

2.4.2. Deuxième exemple : Le problème DriverLog[3]

Ce problème appartient lui aussi à ceux des compétitions IPC. Cependant, nous y avons effectué certains changements pour mettre en évidence les contraintes de ressources rechargeables.

Il s'agit d'un certain nombre de camions qui doivent délivrer des objets d'un endroit à un autre. Chaque camion est conduit par un conducteur et ceux-ci doivent marcher à pied d'un camion à un autre pour les conduire à la destination voulue.

Pour simplifier le problème, on suppose que chaque camion a une vitesse et une consommation moyenne de carburant indépendantes de la charge qu'il contienne et qu'il puisse se charger en carburant à chaque dépôt d'objets (endroit). En plus, tous les chauffeurs ont la même vitesse de déplacement à pied.

Les actions possibles sont :

a. L'action LOAD-TRUCK

Cette action prend en paramètre un objet *Object*, un camion *Truck* et un endroit *Location*. Il s'agit de charger l'objet *Object* dans le camion *Truck*.

Pour que cette action soit exécutable, il faut que :
- le camion *Truck* et l'objet *Object* soient situés à l'endroit *Location*.
- la capacité libre du camion *Truck* puisse supporter le volume de l'objet *Object.*

Les effets de cette action sont :
- la capacité libre du camion *Truck* diminue selon le volume de l'objet *Object*.
- l'objet *Object* se situe dans le camion *Truck* et non plus à l'endroit *Location*.

Cette action a une durée fixe.

[3] http://planning.cis.strath.ac.uk/competition/domains.html

b. L'action UNLOAD-TRUCK

Cette action prend en paramètre un objet *Object*, un camion *Truck* et un endroit *Location*. Il s'agit de décharger l'objet *Object* du camion *Truck*.

Pour que cette action soit exécutable, il faut que :
- le camion *Truck* contienne l'objet *Object*.
- le camion *Truck* soit situé à l'endroit *Location*.

Les effets de cette action sont :
- la capacité libre du camion *Truck* augmente selon le volume de l'objet *Object*.
- l'objet *Object* se situe à l'endroit *Location* et non plus dans le camion *Truck*.

Cette action a une durée fixe.

c. L'action BOARD-TRUCK

Cette action prend en paramètre un camion *Truck*, un chauffeur *Driver* et un endroit *Location*. Il s'agit d'embarquer le chauffeur *Driver* dans le camion *Truck*.

Pour que cette action soit exécutable, il faut que :
- le camion *Truck* et le chauffeur *Driver* soient situés à l'endroit *Location*.
- le camion *Truck* ne contienne aucun chauffeur à bord.

Les effets de cette action sont :
- le camion *Truck* contient le chauffeur *Driver* à bord.
- le chauffeur *Driver* se situe dans le camion *Truck* et non plus à l'endroit *Location*.

Cette action a une durée fixe.

d. L'action DISEMBARK-TRUCK

Cette action prend en paramètre un camion *Truck*, un chauffeur *Driver* et un endroit *Location*. Il s'agit de débarquer le chauffeur *Driver* du camion *Truck*.

Pour que cette action soit exécutable, il faut que :

- le camion *Truck* soit situé à l'endroit *Location*.
- le camion *Truck* contienne le chauffeur *Driver* à bord.
- le chauffeur *Driver* soit situé dans le camion *Truck*.

Les effets de cette action sont :
- le camion *Truck* ne contient plus aucun chauffeur à bord.
- le chauffeur *Driver* se situe à l'endroit *Location* et non plus dans le camion *Truck*.

Cette action a une durée fixe.

e. L'action DRIVE-TRUCK

Cette action prend en paramètre un camion *Truck*, un endroit de départ *Loc-From* et un endroit cible *Loc-To*. Il s'agit de conduire le camion *Truck* de l'endroit *Loc-From* à l'endroit *Loc-To*.

Pour que cette action soit exécutable, il faut que :
- le camion *Truck* soit situé à l'endroit *Loc-From*.
- le camion *Truck* contienne assez de carburant pour effectuer le déplacement.
- le camion *Truck* contienne un chauffeur à bord.
- il existe un chemin routier entre l'endroit *Loc-From* et l'endroit *Loc-To*.

Les effets de cette action sont :
- le camion *Truck* se situe à l'endroit *Loc-To* et non plus à l'endroit *Loc-From*.
- la quantité de carburant contenue dans le camion *Truck* diminue selon la distance routière entre les deux endroits *Loc-From* et *Loc-To*.

Cette action a une durée en relation avec la distance routière entre les deux endroits *Loc-From* et *Loc-To*

f. L'action WALK

Cette action prend en paramètre un chauffeur *Driver*, un endroit de départ *Loc-From* et un endroit cible *Loc-To*. Il s'agit de déplacer le chauffeur *Driver* de l'endroit *Loc-From* à l'endroit *Loc-To*.

Pour que cette action soit exécutable, il faut que :

- le chauffeur *Driver* soit situé à l'endroit *Loc-From*.
- il existe un chemin à pied entre l'endroit *Loc-From* et l'endroit *Loc-To*.

Cette action a l'effet suivant :

- le chauffeur *Driver* se situe à l'endroit *Loc-To* et non plus à l'endroit *Loc-From*.

Cette action a une durée en relation avec la distance à pied entre les deux endroits *Loc-From* et *Loc-To*

g. L'action REFUEL

Cette action prend en paramètre un camion *Truck* et un endroit *Location*. Il s'agit de recharger le camion *Truck* de carburant jusqu'à sa capacité maximale.

Pour que cette action soit exécutable, il faut que :

- le camion *Truck* soit situé à l'endroit *Location*.
- la quantité de carburant contenue dans le camion *Truck* soit inférieure à la capacité maximale de ce camion.

Cette action a l'effet suivant :

- la capacité en carburant du camion *Truck* devient égale à la capacité maximale de ce camion.

Cette action a une durée fixe.

L'annexe D contient un codage de ce problème en langage PDDL 2.1.

Le tableau 4.5 présente les résultats expérimentaux des effets des modifications effectuées sur le planificateur CPT pour le problème DriverLog avec des ressources rechargeables par assignation (pour le cas du carburant) et par incrémentation (pour le cas de la charge des camions). Dans toutes les instances testées, le nombre de ressources rechargeables par assignation est le même que celui des ressources rechargeables par incrémentation qui est aussi le nombre de camions dans le problème. Le nombre d'objets et le nombre de chauffeurs varient d'une instance à une autre.

Nombre de camions	Nombre de ressources rechargeables	Nombre d'endroits	Temps de recherche moyen dans CPT modifié (en secondes)
2	4	5	1.15
2	4	10	3.32
3	6	10	10.27
3	6	15	20.12
4	8	20	110,04
4	8	25	240,31

Tableau 4.5: Expérimentation du planificateur CPT prenant en compte les ressources rechargeables par assignation et par incrémentation dans le problème DriverLog

Le tableau 4.6 présente les résultats expérimentaux de l'effet des modifications effectuées sur le planificateur CPT pour le problème DriverLog sans ressources. Pour les objets à transporter, nous avons considéré que chaque camion ne prend qu'un seul objet à la fois.

Nous avons mis en comparaison le temps moyen de recherche écoulé par la version initiale de CPT et le temps moyen de recherche écoulé par la version modifiée du même planificateur.

Nombre de camions	Nombre d'endroits	Temps de recherche moyen dans CPT initial (en secondes)	Temps de recherche moyen dans CPT modifié (en secondes)
2	5	0.23	0.31
2	10	0.37	0.49
3	10	6.33	7.11
3	15	9.54	12.03
4	20	45.17	50.25
4	25	55.13	59.44

Tableau 4.6: Comparaison entre la version initiale et la version modifiée du planificateur CPT pour le traitement des problèmes de planification temporelle sans ressources dans le problème DriverLog

2.4.3 Analyse des résultats

On remarque qu'après modifications, le planificateur CPT garde toutes ses performances en tant que planificateur temporel optimal parallèle, car il n'y a aucun changement remarquable de résultat dans les problèmes de planification temporelle sans ressources entre la version initiale et la version modifiée (la différence de temps entre les deux versions du planificateur est négligeable).

Les problèmes de planification temporelle avec ressources sont encore plus complexes car ils utilisent à la fois les contraintes temporelles et les contraintes numériques, ce qui fait que

certains problèmes nécessitent un traitement un peu long, en particulier les problèmes où la recharge se fait par incrémentation et dans lesquels certaines actions consommatrices de carburant nécessitent plusieurs actions de recharge successives pour leur procurer la quantité nécessaire.

CONCLUSION

On peut constater que la méthode basée sur les liens de ressources permet d'améliorer le planificateur CPT en lui donnant la capacité de gérer les ressources consommables et renouvelables. En plus, elle permet de conserver les performances du planificateur original pour résoudre les problèmes de planification temporelle sans ressources.

Cependant, la solution basée sur liens de ressources exige certaines restrictions supplémentaires interdisant d'éventuelles solutions valides dans la réalité, notamment la contrainte (C4.2) qui interdit d'exécuter toute action manipulant une ressource r en parallèle avec une action qui génère cette ressource par incrémentation ou par assignation. Par exemple, dans le problème ZenoTravel, en considérant comme ressource le nombre de passagers dans l'avion, il est interdit d'embarquer plus d'un passager à la fois, alors que dans la réalité, ce n'est pas le cas (particulièrement si l'avion possède plusieurs portes d'embarquement).

Dans le chapitre suivant, nous allons établir une autre méthode moins restrictive que celle basée sur les liens de ressources pour permettre au planificateur CPT de gérer les ressources. C'est la méthode basée sur les actions de synchronisation qui permet, non seulement de gérer les ressources rechargeables par incrémentation et par assignation mais en plus, de gérer les actions hybrides qui consomment et génèrent la même ressource à la fois.

CHAPITRE 5

GESTION DES RESSOURCES DANS CPT : METHODE BASEE SUR LES ACTIONS DE SYNCHRONISATION

INTRODUCTION

La méthode basée sur les liens de ressources exposée dans le chapitre précédent a l'inconvénient d'imposer des blocages sur les ressources qui risquent de réfuter des plans valides dans la réalité. Nous avons donné l'exemple du problème ZenoTravel où on interdit d'exécuter plusieurs actions d'embarquement simultanément, alors que dans la vie réelle, ceci est possible, en particulier si l'avion a plusieurs portes d'embarquement.

Dans ce chapitre, nous proposons une solution permettant de gérer les ressources dans un cadre plus général avec affaiblissement des contraintes de blocage des ressources et prise en compte des ressources hybrides.

1. INTRODUCTION DE NOUVEAUX CONCEPTS

1.1. Affaiblissement des contraintes de blocage des ressources

Nous visons à affaiblir, en particulier, les contraintes suivantes (vues dans le chapitre précédent) :

(C4.1) Les actions étant non interruptibles, la quantité $q_{inf}(r, \alpha_1)$ de chaque ressource r nécessaire pour l'exécution de l'action α est bloquée pour cette action tout au long de l'exécution de celle-ci.

(C4.2) Si une action α_1 génère une ressource r (en incrémentant sa capacité ou en assignant celle-ci à une valeur fixe), la ressource est bloquée pour l'action α_1 durant son exécution, c'est-à-dire que toute action α_2 consommatrice ou génératrice de la ressource r ne peut être exécutée dans l'intervalle de temps d'exécution de l'action α_1.

Nous avons constaté qu'il existe des plans réalisables dans la réalité qui ne respectent pas ces deux contraintes. Par exemple, dans le problème ZenoTravel, il y a la ressource « nombre de passagers dans l'avion » qui est incrémentée avec l'embarquement d'un passager dans l'avion. Si celui-ci possède plusieurs portes, il est possible, dans la réalité, d'exécuter

plusieurs actions d'embarquement simultanées à travers les différentes portes de l'avion si on ne dépasse pas la capacité maximale de passagers embarqués. Cependant, cela est interdit par les règles de blocage des ressources. Pour cette raison, notre objectif est de remplacer ces contraintes par des contraintes moins restrictives.

Nous rappelons que, dans l'ordonnancement et la planification temporelle, deux actions α_1 et α_2 sont reliées par une relation de parallélisme (notée $\alpha_1 \parallel \alpha_2$) si et seulement si α_1 et α_2 s'exécutent simultanément pendant au moins une unité de temps.

Nous allons maintenant définir les contraintes qui doivent être respectées par les actions pouvant s'exécuter en parallèle : les d'actions $\alpha_1,\ldots, \alpha_n$ peuvent être exécutées en parallèle sans générer un conflit sur les ressources si elles respectent les contraintes suivantes :

(C5.1) Dans l'ensemble contenant les n actions, il n'existe aucun sous-ensemble d'actions mutex.

(C5.2) Les ressources sont disponibles en quantité valable pour permettre l'exécution des n actions dans tout ordre, c'est-à-dire que toute action peut s'exécuter avant, après ou en même temps que les autres actions.

(C5.3) L'effet total de l'exécution des n actions sur la disponibilité de la ressource r ne change pas quel que soit l'ordre dans lequel on a exécuté ces actions.

Remarque 5.1 :

Dans ce qui suit, la notion d'action qui génèrent une ressource r par assignation sera généralisée et on parlera désormais d'actions qui assignent la disponibilité de r à une valeur fixe (elles peuvent donc générer la ressource r ou la consommer).

Remarque 5.2:

D'après la contrainte (C5.3), nous pouvons déduire que les actions d'assignation de la disponibilité de la ressource r sont mutuellement exclusives (mutex) avec toutes les actions qui consomment ou génèrent cette ressource.

Preuve :

Pour deux actions α et α' où α assigne la ressource r à une quantité q, on peut constater le résultat suivant :

si la quantité de la ressource r dans le réservoir ne permet pas d'exécuter les actions α et α' dans tout ordre alors ces deux actions sont en mutex.

Dans le cas contraire on a :

- si α' est une action qui consomme r, l'effet de l'exécution des deux actions α et α' sur la ressource r change suivant l'ordre dans lequel ces deux actions ont été exécutées : si on exécute α après α' la disponibilité de r sera égale à q alors que si on exécute α avant α' la disponibilité de r sera égale à $q- q_{dec}(r, \alpha') < q$.
- si α' est une action qui génère r par incrémentation, l'effet de l'exécution des deux actions α et α' sur la ressource r change suivant l'ordre dans lequel ces deux actions ont été exécutées : si on exécute α après α' la disponibilité de r sera égale à q alors que si on exécute α avant α' la disponibilité de r sera égale à $q+ q_{inc}(r, \alpha') > q$.
- si α' est une action qui assigne la disponibilité de r à la valeur q', l'effet de l'exécution des deux actions α et α' sur la ressource r change suivant l'ordre dans lequel ces deux actions ont été exécutées : si on exécute α après α' la disponibilité de r sera égale à q alors que si on exécute α avant α' la disponibilité de r sera égale à q'.

D'où on peut conclure que les actions qui assignent la disponibilité d'une ressource r sont mutex avec toutes les autres actions manipulant cette même ressource r.

1.2. Notion d'action hybride

Les actions hybrides sont des actions à la fois consommatrices et génératrices d'une même ressource r, c'est-à-dire qu'elles consomment une certaine quantité de r (elles nécessitent donc un niveau minimal de la disponibilité de cette ressource) et elles génèrent une certaine quantité de cette ressource (elles peuvent donc exiger un niveau maximal au réservoir de la ressource r) au cours du même intervalle de temps.

Les actions hybrides peuvent représenter par exemple le fonctionnement du moteur d'une voiture qui, pour démarrer, consomme une quantité d'énergie fournie par la batterie d'accumulateur et, en fonctionnant, recharge celle-ci.

1.3. Notion d'action de synchronisation

Pour pouvoir mettre en œuvre une méthode permettant de gérer ces nouvelles contraintes, nous avons besoin d'introduire des actions de synchronisation.

Définition 5.1: Actions de synchronisation

Les actions de synchronisation relatives à une ressource r sont définies comme des actions fictives de durée nulle qui ne consomment aucune ressource et qui précèdent ou succèdent un sous-ensemble $\mathcal{A}'_i \subseteq \mathcal{A}$ d'actions de recharge par incrémentation et/ou de consommation de r telles que la disponibilité de la ressource r permet de les exécuter toutes en parallèle (c'est-à-dire que les contraintes C5.2 et C5.3 sont respectées).

Donc, chaque couple d'actions de synchronisation successives (S_i, S_{i+1}) relatives à une ressource r (avec $S_i<S_{i+1}$) délimite un sous-ensemble $\mathcal{A}'_i$ d'actions pouvant être exécutées en parallèle (du point de vue disponibilité de la ressource r) qui consomment ou génèrent par incrémentation cette ressource.

On a : $(\forall \alpha \in \mathcal{A}'_i / \mathit{InPlan}(\alpha)), S_i< \alpha <S_{i+1})$

Et on note : $S_i=S_{bef}(r,\mathcal{A}'_i)$ et $S_{i+1}=S_{aft}(r,\mathcal{A}'_i)$.

Cette contrainte de précédence est assurée par un jeu de fluents :

- le fluent *Prod*(r) est généré par l'action de synchronisation $S_{bef}(r,\mathcal{A}'_i)$, il est demandé comme précondition par toutes les actions de $\mathcal{A}'_i$ et est retiré par l'action de synchronisation $S_{aft}(r,\mathcal{A}'_i)$ après son exécution.

- un fluent *Mod*(r) est généré par chaque action de $\mathcal{A}'_i$ et il est demandé par l'action de synchronisation $S_{aft}(r,\mathcal{A}'_i)$ qui le retire après son exécution.

Le fluent *Prod*(r), généré par l'action de synchronisation S_i et demandé par toute action α de $\mathcal{A}'_i$ qui consomme la ressource r, assure le parallélisme entre ces actions. En effet, l'action

$\alpha \in \mathcal{A}'_i$ consomme le fluent *Prod*(r) sans le supprimer, ce qui permet aux autres actions de $\mathcal{A}'_i$ qui consomment *Prod*(r) de s'exécuter en parallèle avec α.

Toutes les actions qui consomment le fluent *Prod*(r) doivent s'exécuter avant l'action de synchronisation qui génère ce fluent, c'est-à-dire l'action S_i.

Tous les fluents *Prod*(r) sont retirés par l'action S_{i+1}.

Le fluent *Mod*(r), généré par chaque action de $\mathcal{A}'_i$ qui consomme r et demandé puis supprimé par l'action de synchronisation S_{i+1}, assure l'exécution de S_{i+1} après l'exécution de toutes les actions de $\mathcal{A}'_i$.

Ce mécanisme permet d'ordonner les actions d'une manière à ce que chaque action de $\mathcal{A}'_i$ soit précédée de l'action S_i et suivie de l'action S_{i+1}.

On rappelle qu'une action de synchronisation relative à une ressource r est en mutex avec toutes les autres actions de synchronisation relatives à cette ressource ainsi qu'avec toutes les actions qui génèrent ou consomment la ressource r. Elle retire donc les fluents *Mod*(r) demandés par les actions de synchronisation et générés par les actions de consommation et/ou de production de la ressource.

Le lien causal $\mathcal{S}$(*Mod*(r),S_{i+1}) est formé par toutes les actions qui génèrent le fluent *Mod*(r). Le domaine $\mathcal{D}[\mathcal{S}(Mod(r),S_{i+1})]$ est composé des sous-ensembles d'actions qui génèrent le fluent *Mod*(r).

Une action est exclue du domaine $\mathcal{D}[\mathcal{S}(Mod(r),S_{i+1})]$ si elle est éliminée de tous les éléments de ce domaine.

Remarque 5.3:

Une action qui assigne la disponibilité de la ressource r à une valeur fixe ne peut pas s'exécuter en parallèle avec d'autres actions qui consomment ou génèrent la même ressource. C'est pourquoi elle doit toujours être seule dans un intervalle de temps délimité par deux actions de synchronisation successives.

Pour chaque sous-ensemble $\mathcal{A}'_i \subseteq \mathcal{A}$ d'actions α_1, α_2,…, α_n qui s'exécutent dans l'intervalle de temps limité par des actions de synchronisation successives S_i et S_{i+1} relatives à la ressource r (avec $S_i<S_{i+1}$ et $\nexists$ S_j tel que $S_i<S_j$ et $S_j<S_{i+1}$) on considère les notations suivantes :

$\mathcal{A}'_{ig}$(r) est le sous-ensemble de $\mathcal{A}'_i$ contenant les actions qui génèrent la ressource r.

$\mathcal{A}'_{ic}$(r) est le sous-ensemble de $\mathcal{A}'_i$ contenant les actions qui consomment la ressource r.

On a donc $\mathcal{A}'_i = \mathcal{A}'_{ig}(r) \cup \mathcal{A}'_{ic}(r)$.

Remarque 5.4:

Les deux ensembles $\mathcal{A}'_{ic}$(r) et $\mathcal{A}'_{ig}$(r) ne sont pas nécessairement disjoints car ils peuvent contenir des actions hybrides qui appartiennent aux deux ensembles simultanément.

Exemple de cas valide de synchronisation

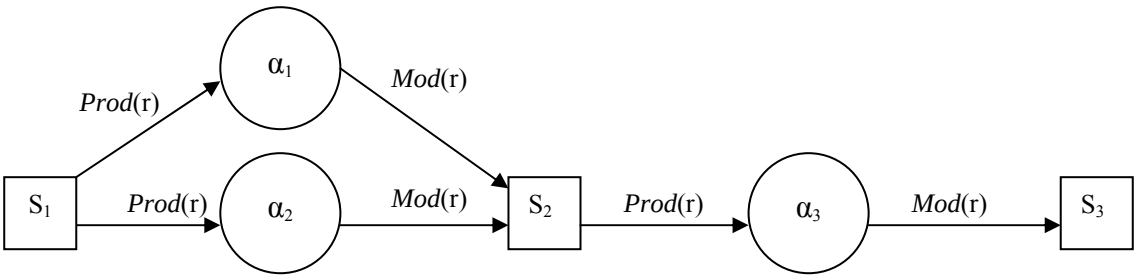

Figure 5.1: Cas valide de synchronisation

La disponibilité de r au moment de l'exécution de S_2 est calculée par la formule suivante :

$Q(S_2) = Q(S_1) + \sum_{\alpha \in A'g(r)} (q_{inc}(r, \alpha)) - \sum_{\alpha \in A'c(r)} (q_{dec}(r, \alpha))$.

Ce qui peut être généralisé pour les n actions de synchronisation d'un plan partiel :

$\forall i \in \{0,\ldots,n-1\}$, $Q(S_{i+1}) = Q(S_i) + \sum_{\alpha \in \mathcal{A}ig(r)} (q_{inc}(r, \alpha)) - \sum_{\alpha \in \mathcal{A}ic(r)} (q_{dec}(r, \alpha))$.

Exemples de cas invalides de synchronisation

1er cas

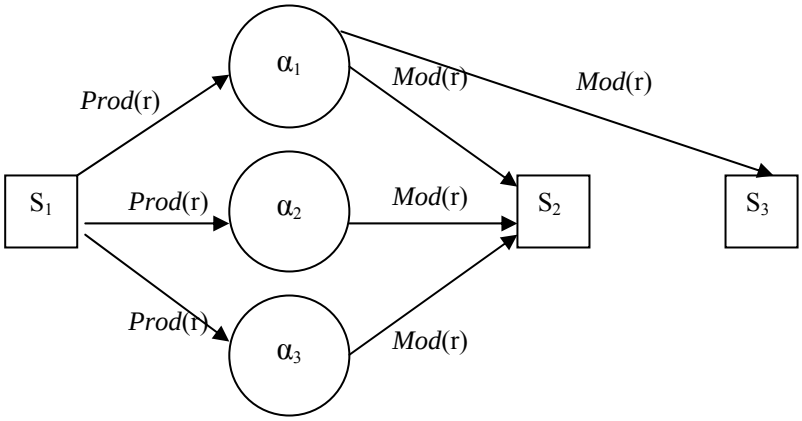

Figure 5.2: Cas invalide de synchronisation

Ce cas ne peut pas être réalisé par le système car l'action de synchronisation S_2 va retirer le fluent *Mod*(r), ce qui empêche la création d'un lien entre l'action α_1 et l'action de synchronisation S_3.

2ème cas

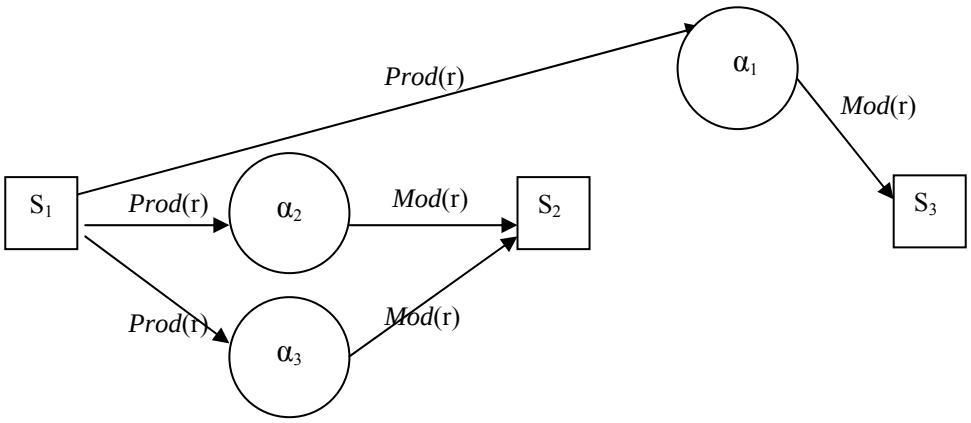

Figure 5.3: Un autre cas invalide de synchronisation

Même si l'action S_2 ne peut pas retirer le fluent *Mod*(r) généré par l'action α_1, ce cas ne peut pas être réalisé car l'action S_2 va retirer les fluents *Prod*(r) et en particulier celui utilisé par l'action α_1. Ceci garantit le fait qu'une action α doit être exécutée entre deux actions de synchronisation successives dont l'une va générer le fluent *Prod*(r) demandé par l'action α et l'autre va consommer le fluent *Mod(*r) généré par cette action.

1.4. Parallélisme entre actions de recharge et actions de consommation

Proposition 5.1

Soit un ensemble $\mathcal{A}'_i$ contenant n actions génératrices de r par incrémentation et m actions consommatrices de r. Les conditions suivantes sont nécessaires et suffisantes pour garantir que toutes les actions de $\mathcal{A}'_i$ puissent être exécutées en parallèle dans tout ordre d'exécution :

- l'ensemble $\mathcal{A}'_i$ ne contient pas d'actions mutex.

- en choisissant les deux actions de synchronisation $S_i=S_{bef}(r,\mathcal{A}'_i)$ et $S_{i+1}=S_{aft}(r,\mathcal{A}'_i)$, on a :

$$\text{Max}_{\alpha 1 \in A'ic(r)} \left(q_{inf}(r, \alpha_1) + \sum_{\alpha \in A'ic(r)\setminus\{\alpha 1\}} q_{dec}(r, \alpha)\right) \leq Q(S_i) \leq \text{Min}_{\alpha 1 \in A'ig(r)} \left(q_{sup}(r, \alpha 1) - \sum_{\alpha \in A'ig(r)\setminus\{\alpha 1\}} q_{inc}(r, \alpha)\right).$$

Preuve:

Nécessité des conditions :

Il est évident que les actions qui s'exécutent en parallèle ne doivent pas être des actions mutex au sens de la planification temporelle.

Supposons qu'il existe une action $\alpha_i \in \mathcal{A}'_{ic}(r)$ telle que :

$$Q(S_i) < q_{inf}(r, \alpha_i) + \sum_{\alpha \in A'ic(r)\setminus\{\alpha_i\}} q_{dec}(r, \alpha)$$

Si on commence par exécuter toutes les actions de $\mathcal{A}'_{ic}(r)\setminus\{\alpha_i\}$ avant les actions de $\mathcal{A}'_{ig}(r)$, on n'aura pas assez de ressources pour exécuter l'action α_i. Cet ordre est donc invalide ce qui permet de conclure la nécessité de la condition :

$$Q(S_i) \geq \text{Max}_{\alpha 1 \in A'ic(r)} \left(q_{inf}(r, \alpha_1) + \sum_{\alpha \in A'ic(r)\setminus\{\alpha 1\}} q_{dec}(r, \alpha)\right).$$

Supposons maintenant qu'il existe une action $\alpha_j \in \mathcal{A}'_{ig}(r)$ telle que :

$$Q(S_i) > q_{sup}(r, \alpha_j) - \sum_{\alpha \in A'ig(r)\setminus\{\alpha j\}} (q_{inc}(r, \alpha))$$

Si on commence par exécuter toutes les actions de $\mathcal{A}'_{ig}(r)\setminus\{\alpha_j\}$ avant les actions de $\mathcal{A}'_{ic}(r)$, la disponibilité de la ressource sera trop grande pour pouvoir exécuter l'action α_j. Cet ordre est donc invalide ce qui permet de conclure la nécessité de la condition :

$$Q(S_i) \leq \text{Min}_{\alpha 1 \in A'ig(r)} \left(q_{sup}(r, \alpha_1) - \sum_{\alpha \in A'ig(r)\setminus\{a1\}} q_{inc}(r, \alpha)\right).$$

La condition $\text{Max}_{\alpha 1 \in A'ic(r)} (q_{inf}(r, \alpha_1) + \sum_{\alpha \in A'ic(r)\setminus\{\alpha 1\}} q_{dec}(r, \alpha)) \leq Q(S_i) \leq \text{Min}_{\alpha 1 \in A'ig(r)} (q_{sup}(r, \alpha_1) - \sum_{\alpha \in A'ig(r)\setminus\{\alpha 1\}} q_{inc}(r, \alpha))$ est donc nécessaire.

Suffisance des conditions :

On suppose avoir un ensemble $\mathcal{A}'_i$ d'actions dont certaines sont consommatrices et d'autres sont génératrices de la ressource r et qui respectent les conditions suivantes :

- l'ensemble $\mathcal{A}'_i$ ne contient pas d'actions mutex.

- $\text{Max}_{\alpha 1 \in A'ic(r)} (q_{inf}(r, \alpha_1) + \sum_{\alpha \in A'ic(r)\setminus\{\alpha 1\}} q_{dec}(r, \alpha)) \leq Q(S_i) \leq \text{Min}_{\alpha 1 \in A'ig(r)} (q_{sup}(r, \alpha_1) - \sum_{\alpha \in A'ig(r)\setminus\{\alpha 1\}} q_{inc}(r, \alpha))$.

Le but est de montrer que tout ordre d'exécution des actions est possible.

Soit une action quelconque $\alpha_i \in \mathcal{A}'_{ic}(r)$. Supposons qu'il y a un ordre dans lequel la ressource r est insuffisante pour exécuter cette action α_i.

L'ordre dans lequel la disponibilité de la ressource r atteint son niveau minimal est de commencer par exécuter toutes les actions de $\mathcal{A}'_{ic}(r)\setminus\{\alpha_i\}$ avant toute action de $\mathcal{A}'_{ig}(r)$ et dans cet ordre, la disponibilité de r après exécution de toutes les actions de $\mathcal{A}'_{ic}(r)/\{\alpha_i\}$ est $Q(S_i) - \sum_{\alpha \in A'ic(r)\setminus\{\alpha i\}} q_{dec}(r, \alpha)$.

Donc l'action α_i n'aura pas une disponibilité de la ressource r suffisante pour être exécutée si $Q(S_i) - \sum_{\alpha \in A'ic(r)\setminus\{\alpha i\}} q_{dec}(r, \alpha) < q_{inf}(r, \alpha_i)$, ce qui est impossible puisque $Q(S_i) \geq \text{Max}_{\alpha 1 \in A'ic(r)} (q_{inf}(r, \alpha_1) + \sum_{\alpha \in A'ic(r)\setminus\{\alpha 1\}} q_{dec}(r, \alpha))$.

D'où le but recherché pour ce cas.

Soit maintenant une action quelconque $\alpha_j \in \mathcal{A}'_{ig}(r)$. Supposons qu'il y a un ordre dans lequel la disponibilité de la ressource r est trop grande pour pouvoir exécuter cette action.

L'ordre dans lequel la disponibilité des ressources atteint son niveau maximal est de commencer par exécuter toutes les actions de $\mathcal{A}'_{ig}(r)\setminus\{\alpha_j\}$ avant toute action de $\mathcal{A}'_{ic}(r)$ et dans cet ordre, la disponibilité de r après exécution de toutes les actions de $\mathcal{A}'_{ig}(r)\setminus\{\alpha_j\}$ est $Q(S_i) + \sum_{\alpha \in A'ig(r)\setminus\{\alpha j\}} q_{inc}(r, \alpha)$.

Donc l'action α_j rencontrera un niveau trop élevé de la ressource r si $Q(S_i) + \sum_{\alpha \in A'ig(r)/\{\alpha j\}} q_{inc}(r, \alpha) > q_{sup}(r, \alpha_j)$, ce qui est impossible puisque $Q(S_i) \leq \text{Min}_{\alpha 1 \in A'ig(r)} (q_{sup}(r, \alpha_1) - \sum_{\alpha \in A'ig(r)/\{\alpha 1\}} q_{inc}(r, \alpha))$.

D'où le but recherché pour ce cas.

D'après les deux conclusions obtenues, on constate que toute action de $\mathcal{A}'_i$ peut être exécutée dans n'importe quel ordre par rapport au reste des actions de $\mathcal{A}'_i$ puisque nous avons :

$\mathcal{A}'_i = \mathcal{A}'_{ig}(r) \cup \mathcal{A}'_{ic}(r)$.

1.4.1. Parallélisme entre actions de consommation

Pour chaque action α du sous-ensemble $\mathcal{A}'_{ic}(r)$, la disponibilité minimale requise de r, notée $q_{inf}(r, \alpha)$, doit être inférieure à la différence entre la disponibilité de la ressource r à « l'exécution »[4] de l'action de synchronisation S_i, notée $Q(S_i)$ et les quantités de r consommées par toutes les autres actions de $\mathcal{A}'_{ic}(r)$.

On a donc :

$\forall \alpha \in \mathcal{A}'_{ic}(r), q_{inf}(r, \alpha) \leq Q(S_i) - \sum_{\alpha 1 \in A'c(r)\setminus\{\alpha\}} (q_{dec}(r, \alpha_1))$.

Exemple d'actions de consommation

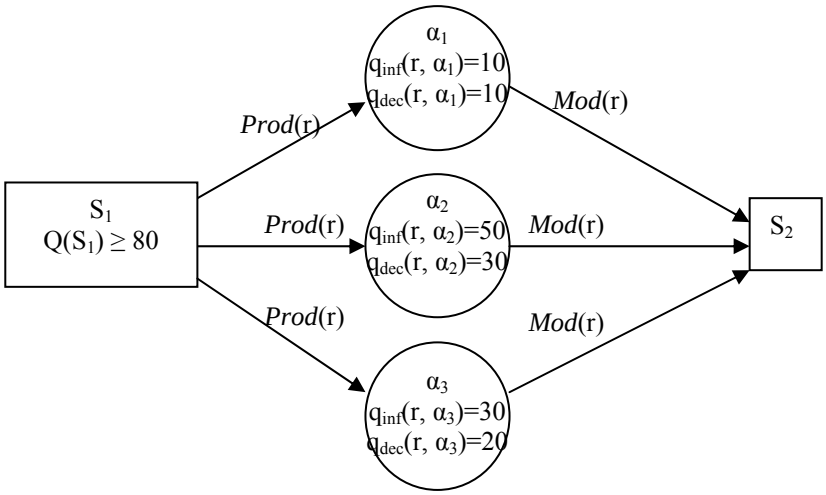

Figure 5.4: Synchronisation pour des actions consommatrices

[4] Les actions de synchronisation sont des actions fictives qui n'ont pas de durée d'exécution, on ne peut donc pas parler d'exécution au sens concret mais il s'agit de mettre l'action considérée en place dans une étape bien précise du plan partiel

Dans cet exemple, on remarque que l'action de synchronisation S_1 demande une quantité $Q(S_1)=80$ pour que les actions de l'ensemble $\mathcal{A}'_1$ soient exécutées avec succès. Cette quantité sera fournie par une ou plusieurs actions qui vont produire la ressource r soit par incrémentation soit par assignation. Ces actions vont être exécutées en totalité avant l'action S_1.

1.4.2. Parallélisme entre actions de recharge par incrémentation

Pour chaque action α du sous-ensemble $\mathcal{A}'_{ig}(r)$, la quantité maximale autorisée de r, notée $q_{sup}(r, \alpha)$, doit être supérieure à la somme des quantités de r produites par toutes les autres actions de $\mathcal{A}'_{ig}(r)$ et de la disponibilité de la ressource r à l'exécution de l'action de synchronisation S_i, notée $Q(S_i)$.

On a donc:

$\forall \alpha \in \mathcal{A}'_{ig}(r), q_{sup}(r, \alpha) \geq Q(S_i) + \sum_{\alpha 1 \in A'ig(r)\setminus\{\alpha\}} (q_{inc}(r, \alpha_1))$.

Exemple d'actions génératrices par incrémentation

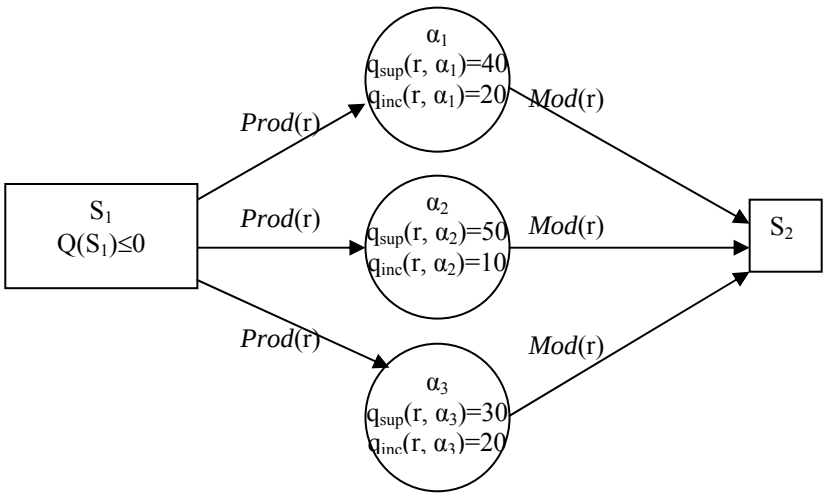

Figure 5.5: Synchronisation pour des actions génératrices par incrémentation

Dans cet exemple, on remarque que la quantité de r disponible avant l'exécution de l'action de synchronisation S_1 doit être nulle pour que les actions de l'ensemble $\mathcal{A}'_1$ soient exécutées avec succès.

Exemple d'actions de consommation et de recharge par incrémentation

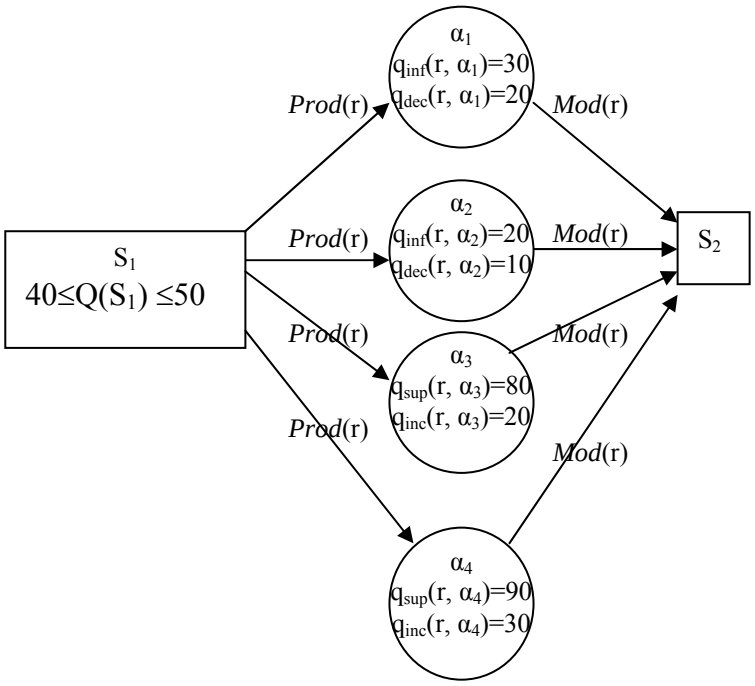

Figure 5.6: Synchronisation pour des actions consommatrices et génératrices

Dans cet exemple, on voit bien qu'on doit avoir $40 \leq Q(S_1) \leq 50$ pour pouvoir exécuter les quatre actions α_1, α_2, α_3 et α_4 en parallèle.

1.4.3. Parallélisme entre actions hybrides

Deux actions hybrides peuvent être exécutées en parallèle si la condition de disponibilité de la ressource r à l'instant d'exécution de l'action de synchronisation est vérifiée. C'est-à-dire :

$$\begin{cases} Q(S_i) \leq \text{Min}_{\alpha 1 \in A'ig(r)} \left(q_{sup}(r, \alpha_1) - \sum_{\alpha \in A'ig(r) \setminus \{\alpha 1\}} (q_{inc}(r, \alpha)) \right) \\ Q(S_i) \geq \text{Max}_{\alpha 1 \in A'ic(r)} \left(q_{inf}(r, \alpha_1) + \sum_{\alpha \in A'ic(r) \setminus \{\alpha 1\}} (q_{dec}(r, \alpha)) \right) \end{cases}$$

Exemple d'actions hybrides

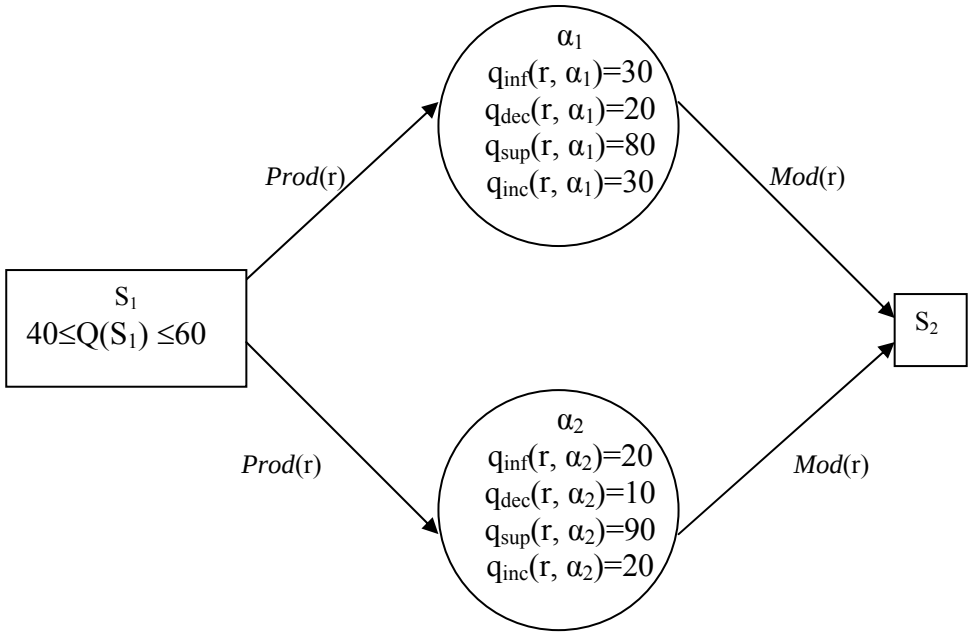

Figure 5.7: Synchronisation pour des actions hybrides

Dans les quatre cas donnés ci-dessus, on remarque qu'avec la condition :

$$\begin{cases} Q(S_1) \leq \text{Min}_{\alpha 1 \in A'1g(r)} \left(q_{sup}(r, \alpha_1) - \sum_{\alpha \in A'1g(r) \setminus \{\alpha 1\}} (q_{inc}(r, \alpha))\right) \\ Q(S_1) \geq \text{Max}_{\alpha 1 \in A'1c(r)} \left(q_{inf}(r, \alpha_1) + \sum_{\alpha \in A'1c(r) \setminus \{\alpha 1\}} (q_{dec}(r, \alpha))\right) \end{cases}$$

on peut toujours garantir la possibilité d'exécution des actions dans tout ordre et en conséquence le parallélisme est autorisé.

1.5. Détermination des bornes de $Q(S_i)$

1.5.1. Contrainte nécessaire pour le parallélisme

La contrainte

$$\begin{cases} Q(S_i) \leq \text{Min}_{\alpha 1 \in A'ig(r)} \left(q_{sup}(r, \alpha_1) - \sum_{\alpha \in A'ig(r) \setminus \{\alpha 1\}} (q_{inc}(r, \alpha))\right) \\ Q(S_i) \geq \text{Max}_{\alpha 1 \in A'ic(r)} \left(q_{inf}(r, \alpha_1) + \sum_{\alpha \in A'ic(r) \setminus \{\alpha 1\}} (q_{dec}(r,\alpha))\right) \end{cases}$$

permet de limiter la disponibilité de la ressource r pour l'action de synchronisation S_i dans un intervalle $[B_{inf}(Q(S_i)),B_{sup}(Q(S_i))]$. En effet, pour que cette contrainte soit satisfaite, il faut que :

$$\begin{cases} B_{inf}(Q(S_i)) \geq \text{Max}_{\alpha 1 \in A'ic(r)} \left(q_{inf}(r, \alpha_1) + \sum_{\alpha \in A'ic(r)\setminus\{\alpha 1\}} (q_{dec}(r, \alpha))\right) & \text{si } \mathcal{A}'_{ic}(r) \neq \varnothing \\ B_{inf}(Q(S_i)) = -\infty \text{ sinon} \end{cases}$$

Et

$$\begin{cases} B_{sup}(Q(S_i)) \leq \text{Min}_{\alpha 1 \in A'ig(r)} \left(q_{sup}(r, \alpha_1) - \sum_{\alpha \in A'ig(r)\setminus\{\alpha 1\}} (q_{inc}(r, \alpha))\right) \text{ si } \mathcal{A}'_{ig}(r) \neq \varnothing \\ B_{sup}(Q(S_i)) = +\infty \text{ sinon} \end{cases}$$

1.5.2. Propagation de la disponibilité de la ressource

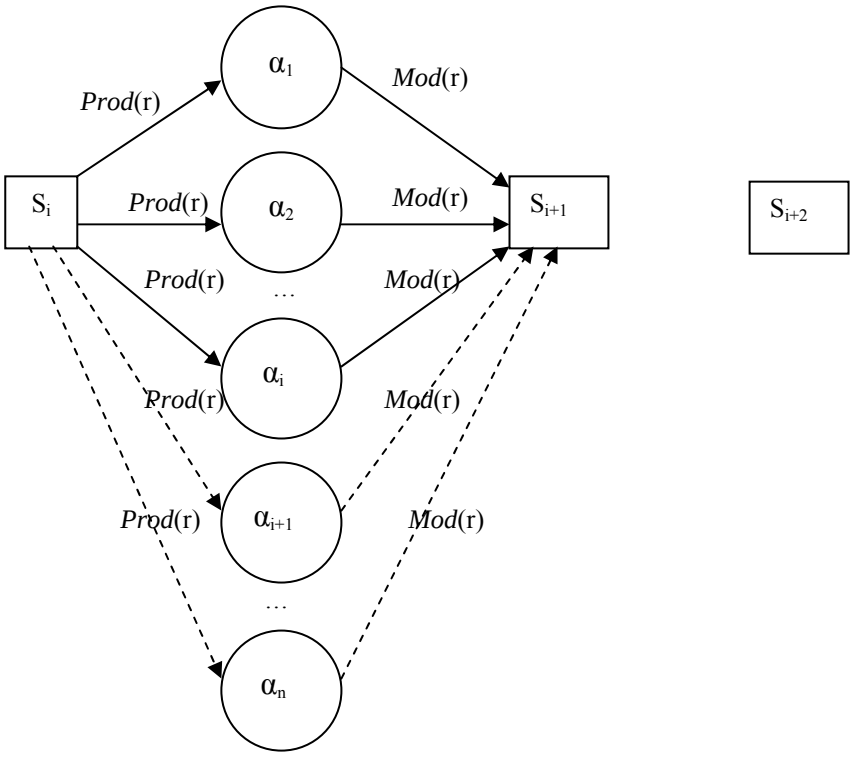

Figure 5.8: Synchronisation et disponibilité

Pour définir la deuxième contrainte, nous utilisons les notations suivantes :

- $Q_{dec}(S_i,S_{i+1})$ représente la quantité de ressources décrémentée entre S_i et S_{i+1}
- $Q_{inc}(S_i,S_{i+1})$ représente la quantité de ressources décrémentée entre S_i et S_{i+1}

La quantité $Q_{dec}(S_i,S_{i+1})$ appartient à l'intervalle $[B_{inf}(Q_{dec}(S_i,S_{i+1})),B_{sup}(Q_{dec}(S_i,S_{i+1}))]$.

Il est à noter que la borne $B_{inf}(Q_{dec}(S_i,S_{i+1}))$ correspond à la somme des décrémentations de la ressource r pour les actions à exécuter entre S_i et S_{i+1} qui sont introduites dans la plan partiel (dans le schéma, elles sont liées par des flèches continues), c'est-à-dire la consommation réelle de la ressource r entre S_i et S_{i+1} et la borne $B_{sup}(Q_{dec}(S_i,S_{i+1}))$ correspond à la somme des décrémentations de la ressource r pour toutes les actions qui peuvent être exécutées entre S_i et S_{i+1} même celles qui ne font pas encore partie du plan partiel.

La quantité $Q_{inc}(S_i,S_{i+1})$ appartient à l'intervalle $[B_{inf}(Q_{inc}(S_i,S_{i+1})),B_{sup}(Q_{inc}(S_i,S_{i+1}))]$.

De façon analogue, la borne $B_{inf}(Q_{inc}(S_i,S_{i+1}))$ correspond à la somme des incrémentations de la ressource r pour les actions entre S_i et S_{i+1} qui sont introduites dans le plan partiel et la borne $B_{sup}(Q_{inc}(S_i,S_{i+1}))$ correspond à la somme des incrémentations de la ressource r de toutes les actions qui peuvent être exécutées entre S_i et S_{i+1}.

La disponibilité de la ressource r au moment de l'exécution de l'action de synchronisation S_{i+1} notée $Q(S_{i+1})$ est donnée par la formule suivante :

$Q(S_{i+1}) = Q(S_i) - Q_{dec}(S_i,S_{i+1}) + Q_{inc}(S_i,S_{i+1})$.

En effet, la différence entre $Q(S_{i+1})$ et $Q(S_i)$ correspond à la différence entre la quantité de ressource générée entre S_i et S_{i+1} représentée par $Q_{inc}(S_i,S_{i+1})$ et la quantité de ressource consommée entre S_i et S_{i+1} représentée par $Q_{dec}(S_i,S_{I+1})$.

Grâce aux bornes de $Q(S_i)$, $Q_{dec}(S_i,S_{i+1})$ et $Q_{inc}(S_i,S_{i+1})$ on a :

$$\begin{cases} Q(S_{i+1}) \le B_{sup}(Q(S_i)) - B_{inf}(Q_{dec}(S_i,S_{i+1})) + B_{sup}(Q_{inc}(S_i,S_{i+1})) \\ Q(S_{i+1}) \ge B_{inf}(Q(S_i)) - B_{sup}(Q_{dec}(S_i,S_{i+1})) + B_{inf}(Q_{inc}(S_i,S_{i+1})) \end{cases}$$

D'où on peut déduire que :

$$\begin{cases} B_{inf}(Q(S_{i+1})) \ge B_{inf}(Q(S_i)) - B_{sup}(Q_{dec}(S_i,S_{i+1})) + B_{inf}(Q_{inc}(S_i,S_{i+1})) \\ B_{sup}(Q(S_{i+1})) \le B_{sup}(Q(S_i)) - B_{inf}(Q_{dec}(S_i,S_{i+1})) + B_{sup}(Q_{inc}(S_i,S_{i+1})) \end{cases}$$

De même on a: $Q(S_{i+2}) = Q(S_{i+1}) - Q_{dec}(S_{i+1},S_{i+2}) + Q_{inc}(S_{i+1},S_{i+2})$

D'où: $Q(S_{i+1}) = Q(S_{i+2}) + Q_{dec}(S_{i+1},S_{i+2}) - Q_{inc}(S_{i+1},S_{i+2})$

Grâce aux bornes de $Q(S_{i+2})$, $Q_{dec}(S_{i+1},S_{i+2})$ et $Q_{inc}(S_{i+1},S_{i+2})$ on a :

$$\begin{cases} Q(S_{i+1}) \leq B_{sup}(Q(S_{i+2})) + B_{sup}(Q_{dec}(S_{i+1},S_{i+2})) - B_{inf}(Q_{inc}(S_{i+1},S_{i+2})) \\ Q(S_{i+1}) \geq B_{inf}(Q(S_{i+2})) + B_{inf}(Q_{dec}(S_{i+1},S_{i+2})) - B_{sup}(Q_{inc}(S_{i+1},S_{i+2})) \end{cases}$$

D'où on peut déduire que :

$$\begin{cases} B_{inf}(Q(S_{i+1})) \geq B_{inf}(Q(S_{i+2})) + B_{inf}(Q_{dec}(S_{i+1},S_{i+2})) - B_{sup}(Q_{inc}(S_{i+1},S_{i+2})) \\ B_{sup}(Q(S_{i+1})) \leq B_{sup}(Q(S_{i+2})) + B_{sup}(Q_{dec}(S_{i+1},S_{i+2})) - B_{inf}(Q_{inc}(S_{i+1},S_{i+2})) \end{cases}$$

D'après la contrainte du paragraphe précédent, on a :

$$\begin{cases} B_{inf}(Q(S_{i+1})) \geq \text{Max}_{\alpha 1 \in A'(i+1)c(r)} \left(q_{inf}(r, \alpha_1) + \sum_{\alpha \in A'(i+1)c(r)\setminus\{\alpha 1\}} (q_{dec}(r, \alpha))\right) \\ B_{sup}(Q(S_{i+1})) \leq \text{Min}_{\alpha 1 \in A'(i+1)g(r)} \left(q_{sup}(r, \alpha_1) - \sum_{\alpha \in A'(i+1)g(r)\setminus\{\alpha 1\}} (q_{inc}(r, \alpha))\right) \end{cases}$$

On obtient alors $\forall\, i \in \{1,\ldots,n-1\}$:

$$\begin{cases} B_{inf}(Q(S_i)) \geq \text{Max}\Big(B_{inf}(Q(S_{i-1})) - B_{sup}(Q_{dec}(S_{i-1},S_i)) + B_{inf}(Q_{inc}(S_{i-1},S_i)),\ B_{inf}(Q(S_{i+1})) \\ \quad + B_{inf}(Q_{dec}(S_i,S_{i+1})) - B_{sup}(Q_{inc}(S_i,S_{i+1})),\ \text{Max}_{\alpha 1 \in A'ic(r)} \left(q_{inf}(r,\alpha_1) + \sum_{\alpha \in A'ic(r)/\{\alpha 1\}} (q_{dec}(r, \alpha))\right)\Big) \\ B_{sup}(Q(S_i)) \leq \text{Min}\Big(B_{sup}(Q(S_{i-1})) - B_{inf}(Q_{dec}(S_{i-1},S_i)) + B_{sup}(Q_{inc}(S_{i-1},S_i)),\ B_{sup}(Q(S_{i+1})) \\ \quad + B_{sup}(Q_{dec}(S_i,S_{i+1})) - B_{inf}(Q_{inc}(S_i,S_{i+1})),\ \text{Min}_{\alpha 1 \in A'ig(r)} \left(q_{sup}(r, \alpha_1) - \sum_{\alpha \in A'ig(r)/\{\alpha 1\}} (q_{inc}(r, \alpha))\right)\Big) \end{cases}$$

Ce raisonnement est applicable à toute action de synchronisation pour déterminer les bornes de la disponibilité de la ressource à laquelle cette action est liée, sauf pour les deux actions suivantes:

- la première action de synchronisation qui sera précédée par l'action *START* pour laquelle la borne $B_{inf}(Q(S_0))$ et la borne $B_{sup}(Q(S_0))$ sont égales et coïncident avec la disponibilité de la ressource r au démarrage.

- la dernière action qui sera suivie par l'action *END* pour laquelle:

$$\begin{cases} B_{inf}(Q(S_n)) \geq \text{Max}(B_{inf}(Q(S_{n-1})) - B_{sup}(Q_{dec}(S_{n-1},S_n)) + B_{inf}(Q_{inc}(S_{n-1},S_n)), \\ \text{Max}_{\alpha 1 \in A'nc(r)} (q_{inf}(r, \alpha_1) + \Sigma_{\alpha \in A'nc(r) \setminus \{\alpha 1\}} (q_{dec}(r, \alpha)))) \\ B_{sup}(Q(S_n)) \leq \text{Min}(B_{sup}(Q(S_{n-1})) - B_{inf}(Q_{dec}(S_{n-1},S_n)) + B_{sup}(Q_{inc}(S_{n-1},S_n)), \\ \text{Min}_{a1 \in A'g(r)} (q_{sup}(r, \alpha_1) - \Sigma_{\alpha \in A'ng(r) \setminus \{\alpha 1\}} (q_{inc}(r, \alpha)))) \end{cases}$$

2. PROPAGATION DES CONTRAINTES OBTENUES

En ajoutant des actions de synchronisation au problème de planification, on augmente la complexité de sa résolution car celle-ci dépend du nombre d'actions dans le problème. Pour cette raison, dans le but de minimiser le temps de recherche d'un plan valide respectant les nouvelles contraintes, nous avons besoin d'établir un certain nombre de règles de propagation permettant d'éviter des branchements inutiles qui ne vont pas aboutir à des solutions valides.

Ces règles de propagation sont déduites à partir des contraintes obtenues précédemment. La propagation sera répétée récursivement par les mécanismes du planificateur CPT à chaque fois qu'un des évènements suivants sera constaté :

- prise en compte d'une classe d'actions dans le plan partiel,
- fixation des liens *Mod* et *Prod* dans le plan partiel,
- modification des bornes de la disponibilité pour les actions de synchronisation,
- exclusion d'un lien *Prod* ou *Mod*,
- modification du nombre d'instances possibles d'une action.

2.1. Prise en compte d'une classe d'actions dans le plan partiel

Dans ses dernières versions, le planificateur CPT donne la possibilité d'exécuter plusieurs fois la même action dans un même plan. Les actions qui peuvent s'exécuter plusieurs fois s'appellent des actions non canoniques (contrairement aux actions canoniques qui ne s'exécutent qu'une seule fois dans un même plan).

La gestion des actions non canoniques dans le planificateur CPT est possible grâce à un mécanisme qui affecte, pour chaque action non canonique, un type (ou une classe) précis

défini par un ensemble de préconditions, un ensemble d'effets et une durée, et permet de générer un nombre d'instances différentes de ce type qui seront exécutées comme des actions ordinaires. Autrement dit, les instances d'une action non canonique sont des actions similaires qui ont les mêmes préconditions, les mêmes effets et la même durée.

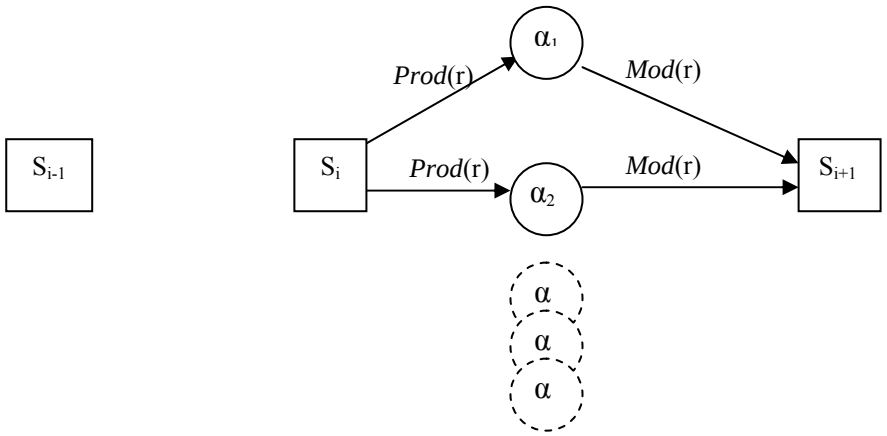

Figure 5.9: Synchronisation et actions multi-instances

La prise en compte d'une action α (ou d'une classe d'actions) dans un plan partiel s'effectue à la phase de prétraitement du planificateur CPT. Il s'agit de trouver, dans ce plan partiel, les étapes dans lesquelles il est possible d'exécuter cette action α (ou cette classe d'actions).

Lors de la prise en compte d'un type d'actions possiblement exécutées dans le plan partiel dans l'intervalle de temps limité par les actions de synchronisation S_i et S_{i+1}, nous devons calculer le nombre maximal d'instances possibles de ce type d'action à ce niveau du plan. Nous pouvons estimer ce nombre grâce à la contrainte :

$$\begin{cases} Q(S_i) \leq \text{Min}_{\alpha 1 \in A'ig(r)} \left(q_{sup}(r, \alpha_1) - \sum_{\alpha \in A'ig(r) \setminus \{\alpha 1\}} (q_{inc}(r, \alpha)) \right) \\ Q(S_i) \geq \text{Max}_{\alpha 1 \in A'ic(r)} \left(q_{inf}(r, \alpha_1) + \sum_{\alpha \in A'ic(r) \setminus \{\alpha 1\}} (q_{dec}(r, \alpha)) \right) \end{cases}$$

En effet, dans le cas général, si l'action α à introduire est une action hybride (à la fois consommatrice et génératrice de la même ressource r) alors l'introduction d'une seule instance de α nécessite les conditions suivantes pour le niveau de la ressource r :

$$\begin{cases} q_{inf}(r, \alpha) + \sum_{\beta \in A'ic(r) \setminus \{\alpha\}} q_{dec}(r, \beta) \leq B_{sup}(S_i) \\ q_{sup}(r, \alpha) - \sum_{\beta \in A'ig(r) \setminus \{\alpha\}} q_{inc}(r, \beta) \geq B_{inf}(S_i) \end{cases}$$

Pour introduire une deuxième instance de l'action α, il faut que :

$$\begin{cases} q_{inf}(r, \alpha) + \sum_{\beta \in A'ic(r)\setminus\{\alpha\}} q_{dec}(r, \beta) + q_{dec}(r, \alpha) \leq B_{sup}(S_i) \\ q_{sup}(r, \alpha) - \sum_{\beta \in A'ig(r)\setminus\{\alpha\}} q_{inc}(r, \beta) - q_{inc}(r, \alpha) \geq B_{inf}(S_i) \end{cases}$$

Et plus généralement, pour introduire une nb^{ième} instance de α il faut que :

$$\begin{cases} q_{inf}(r, \alpha) + \sum_{\beta \in A'ic(r)\setminus\{\alpha\}} q_{dec}(r, \beta) + ((nb-1)*q_{dec}(r, \alpha)) \leq B_{sup}(S_i) \\ q_{sup}(r, \alpha) - \sum_{\beta \in A'ig(r)\setminus\{\alpha\}} q_{inc}(r, \beta) - ((nb-1)*q_{inc}(r, \alpha)) \geq B_{inf}(S_i) \end{cases}$$

Ce qui donne :

$$\begin{cases} nb-1 \leq (B_{sup}(S_i) - q_{inf}(r, \alpha) - \sum_{\beta \in A'ic(r)\setminus\{\alpha\}} q_{dec}(r, \beta)) \,/\, q_{dec}(r, \alpha) \\ nb-1 \leq (B_{inf}(S_i) - q_{sup}(r, \alpha) - \sum_{\beta \in A'ig(r)\setminus\{\alpha\}} q_{inc}(r, \beta)) \,/\, q_{inc}(r, \alpha) \end{cases}$$

D'où la conclusion:

$$\begin{cases} nb \leq ((B_{sup}(S_i) - q_{inf}(r, \alpha) - \sum_{\beta \in A'ic(r)\setminus\{\alpha\}} q_{dec}(r, \beta))/q_{dec}(r, \alpha)) + 1 \\ nb \leq ((B_{inf}(S_i) - q_{sup}(r, \alpha) - \sum_{\beta \in A'ig(r)\setminus\{\alpha\}} q_{inc}(r, \beta)) \,/\, q_{inc}(r, \alpha)) + 1 \end{cases}$$

On peut donc dire que $nb \leq \mathrm{Min}(E((B_{sup}(S_i) - q_{inf}(r, \alpha) - \sum_{\beta \in A'ic(r)\setminus\{\alpha\}} q_{dec}(r, \beta))/q_{dec}(r,\alpha)) + 1,\ E((B_{inf}(S_i) - q_{sup}(r, \alpha) - \sum_{\beta \in A'ig(r)\setminus\{\alpha\}} q_{inc}(r, \beta))/q_{inc}(r, \alpha)) + 1)$

où la fonction E est la fonction partie entière.

L'entier $\mathrm{Min}(E((B_{sup}(S_i) - q_{inf}(r, \alpha) - \sum_{\beta \in A'ic(r)\setminus\{\alpha\}} q_{dec}(r, \beta))/q_{dec}(r,\alpha)) + 1,\ E((B_{inf}(S_i) - q_{sup}(r, \alpha) - \sum_{\beta \in A'ig(r)\setminus\{\alpha\}} q_{inc}(r, \beta))/q_{inc}(r, \alpha)) + 1)$ peut être donc considéré comme un majorant de nb.

Remarque 5.5

Cette contrainte valable pour les actions hybrides peut aussi être appliquée aux actions consommatrices et aux actions génératrices de r :

- pour les actions génératrices de r, l'entier $E((B_{inf}(S_i) - q_{sup}(r, \alpha) - \sum_{\beta \in A'ig(r)\setminus\{\alpha\}} q_{inc}(r,\beta))/q_{inc}(r,\alpha)) + 1$ constitue un majorant de nb.

- Pour les actions consommatrices, l'entier $E((B_{sup}(S_i) - q_{inf}(r, \alpha) - \sum_{\beta \in A'ic(r)\setminus\{\alpha\}} q_{dec}(r,\beta))/q_{dec}(r,\alpha)) + 1$ constitue un majorant de nb.

2.2. Fixation des liens *Mod* et *Prod* d'une action du plan partiel

La deuxième phase dans le prétraitement pour le planificateur CPT est de fixer les liens causaux pour les actions ou les instances prises en compte dans le plan partiel, ce qui permet de préciser la position de l'action ou de l'instance par rapport aux autres actions qui font déjà partie du plan partiel.

La fixation des liens *Mod* et *Prod* pour une action ou une instance permet de préciser l'intervalle de temps délimité par deux actions de synchronisation successives S_i et S_{i+1} dans lequel cette action ou instance est exécutée.

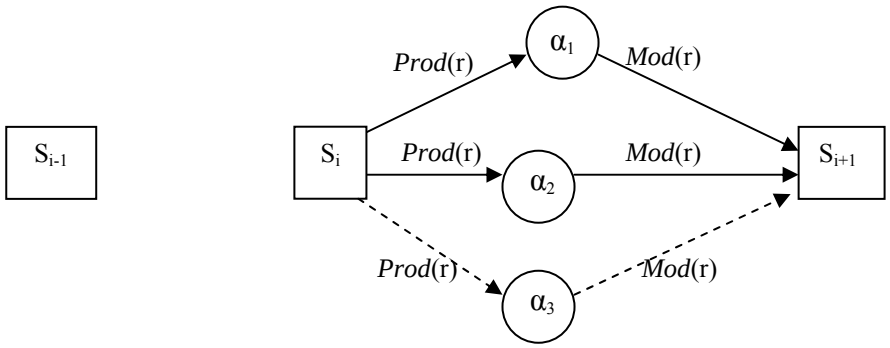

Figure 5.10: Fixation des liens *Prod* et *Mod*

1ère propagation

Pour pouvoir fixer les liens *Mod* et *Prod* de l'action α_3, la contrainte suivante doit être respectée :

$$\begin{cases} Q(S_i) \leq \text{Min}_{\alpha 1 \in A'ig(r)} \left(q_{sup}(r, \alpha_1) - \sum_{\alpha \in A'ig(r)\setminus\{\alpha 1\}} (q_{inc}(r, \alpha))\right) \\ Q(S_i) \geq \text{Max}_{\alpha 1 \in A'ic(r)} \left(q_{inf}(r, \alpha_1) + \sum_{\alpha \in A'ic(r)\setminus\{\alpha 1\}} (q_{dec}(r,\alpha))\right) \end{cases}$$

Il y a éventuellement une mise à jour des bornes de $Q(S_i)$ avec :

$$\begin{cases} B_{inf}^*(Q(S_i)) = \text{Max}(B_{inf}(Q(S_i)), q_{inf}(r, \alpha_3) + \sum_{\alpha \in A'ig(r)\setminus\{\alpha 3\}} q_{dec}(r, \alpha)) \\ B_{sup}^*(Q(S_i)) = \text{Min}(B_{sup}(Q(S_i)), q_{sup}(r, \alpha_3) + \sum_{\alpha \in A'ig(r)\setminus\{\alpha 3\}} q_{inc}(r, \alpha)) \end{cases}$$

où $B_{inf}^*(Q(S_i))$ et $B_{sup}^*(Q(S_i))$ sont respectivement les nouveaux minorant et majorant de S_i.

2ème propagation

En fixant les liens *Mod* et *Prod* d'une action du plan partiel, on fixe l'intervalle d'exécution de cette action entre deux actions de synchronisation. Dans le schéma ci-dessus par exemple, on fixe l'instant d'exécution de α_3 entre les actions de synchronisation S_i et S_{i+1}.

On agit ainsi sur la quantité incrémentée et/ou décrémentée de la ressource r entre les actions de synchronisation S_i et S_{i+1} selon le type de l'action α_3(action consommatrice ou génératrice de r).

En effet, la fixation de l'instant d'exécution d'une action α consommatrice de r entre les deux actions de synchronisation S_i et S_{i+1} augmente de $q_{dec}(r, \alpha)$ la quantité réellement décrémentée de r entre S_i et S_{i+1}. Cette quantité est donnée par le minorant $B_{inf}(Q_{dec}(S_i,S_{i+1}))$.

La mise à jour effectuée influe sur les bornes de $Q(S_i)$ et $Q(S_{i+1})$ puisque nous avons :

$$\begin{cases} B_{sup}(Q(S_{i+1})) \leq B_{sup}(Q(S_i)) - B_{inf}(Q_{dec}(S_i,S_{i+1})) + B_{sup}(Q_{inc}(S_i,S_{i+1})) \\ B_{inf}(Q(S_i)) \geq B_{inf}(Q(S_{i+1})) + B_{inf}(Q_{dec}(S_i,S_{i+1})) - B_{sup}(Q_{inc}(S_i,S_{i+1})) \end{cases}$$

Les bornes mises à jour de $Q(S_i)$ et $Q(S_{i+1})$ sont alors :

$$\begin{cases} B_{sup}^*(Q(S_{i+1})) = \text{Min}(B_{sup}(Q(S_{i+1})), B_{sup}(Q(S_i)) - B_{inf}^*(Q_{dec}(S_i,S_{i+1})) + B_{sup}(Q_{inc}(S_i,S_{i+1}))) \\ B_{inf}^*(Q(S_i)) = \text{Max}\ (B_{inf}(Q(S_i)), B_{inf}(Q(S_{i+1})) + B_{inf}^*(Q_{dec}(S_i,S_{i+1})) - B_{sup}(Q_{inc}(S_i,S_{i+1}))) \end{cases}$$

De même, la fixation de l'instant d'exécution d'une action α génératrice de r entre les deux actions de synchronisation S_i et S_{i+1} augmente de $q_{inc}(r, \alpha)$ la quantité réellement incrémentée de r entre S_i et S_{i+1}. Cette quantité est donnée par le minorant $B_{inf}(Q_{inc}(S_i,S_{i+1}))$, et par suite il y a mise à jour des bornes de $Q(S_i)$ et $Q(S_{i+1})$ puisque nous avons :

$$\begin{cases} B_{inf}(Q(S_{i+1})) \geq B_{inf}(Q(S_i)) - B_{sup}(Q_{dec}(S_i,S_{i+1})) + B_{inf}(Q_{inc}(S_i,S_{i+1})) \\ B_{sup}(Q(S_i)) \leq B_{sup}(Q(S_{i+1})) + B_{sup}(Q_{dec}(S_i,S_{i+1})) - B_{inf}(Q_{inc}(S_i,S_{i+1})) \end{cases}$$

Les nouvelles bornes mises à jour de $Q(S_2)$ et $Q(S_3)$ sont :

$$\begin{cases} B_{inf}^*(Q(S_{i+1})) = \text{Max}(B_{inf}(Q(S_{i+1})), B_{inf}(Q(S_i)) - B_{sup}(Q_{dec}(S_i,S_{i+1})) + B_{inf}^*(Q_{inc}(S_i,S_{i+1}))) \\ B_{sup}^*(Q(S_i)) = \text{Min}(B_{sup}(Q(S_i)), B_{sup}(Q(S_{i+1})) + B_{sup}(Q_{dec}(S_i,S_{i+1})) - B_{inf}^*(Q_{inc}(S_i,S_{i+1}))) \end{cases}$$

Enfin, la fixation de l'instant d'exécution d'une action hybride α entre les deux actions de synchronisation S_i et S_{i+1} augmente de $q_{inc}(r,\alpha)$ le minorant $B_{inf}(Q_{inc}(S_i,S_{i+1}))$ et augmente de $q_{dec}(r, \alpha)$ le minorant $B_{inf}(Q_{dec}(S_i,S_{i+1}))$.

Cette mise à jour des minorants de $Q_{dec}(S_i,S_{i+1})$ et $Q_{inc}(S_i,S_{i+1})$ influe sur les bornes de $Q(S_i)$ et $Q(S_{i+1})$ puisque nous avons :

$$\begin{cases} B_{sup}(Q(S_{i+1})) \le B_{sup}(Q(S_i)) - B_{inf}(Q_{dec}(S_i,S_{i+1})) + B_{sup}(Q_{inc}(S_i,S_{i+1})) \\ B_{inf}(Q(S_i)) \ge B_{inf}(Q(S_{i+1})) + B_{inf}(Q_{dec}(S_i,S_{i+1})) - B_{sup}(Q_{inc}(S_i,S_{i+1})) \\ B_{inf}(Q(S_{i+1})) \ge B_{inf}(Q(S_i)) - B_{sup}(Q_{dec}(S_i,S_{i+1})) + B_{inf}(Q_{inc}(S_i,S_{i+1})) \\ B_{sup}(Q(S_i)) \le B_{sup}(Q(S_{i+1})) + B_{sup}(Q_{dec}(S_i,S_{i+1})) - B_{inf}(Q_{inc}(S_i,S_{i+1})) \end{cases}$$

Les nouvelles bornes mises à jour de $Q(S_i)$ et $Q(S_{i+1})$ sont :

$$\begin{cases} B_{sup}^*(Q(S_{i+1})) = \text{Min}(B_{sup}(Q(S_{i+1})), B_{sup}(Q(S_i)) - B_{inf}^*(Q_{dec}(S_i,S_{i+1})) + B_{sup}(Q_{inc}(S_i,S_{i+1}))) \\ B_{inf}^*(Q(S_i)) = \text{Max}\ (B_{inf}(Q(S_i)), B_{inf}(Q(S_{i+1})) + B_{inf}^*(Q_{dec}(S_i,S_{i+1})) - B_{sup}(Q_{inc}(S_i,S_{i+1}))) \\ B_{inf}^*(Q(S_{i+1})) = \text{Max}(B_{inf}(Q(S_{i+1})), B_{inf}(Q(S_i)) - B_{sup}(Q_{dec}(S_i,S_{i+1})) + B_{inf}^*(Q_{inc}(S_i,S_{i+1}))) \\ B_{sup}^*(Q(S_i)) = \text{Min}(B_{sup}(Q(S_i)), B_{sup}(Q(S_{i+1})) + B_{sup}(Q_{dec}(S_i,S_{i+1})) - B_{inf}^*(Q_{inc}(S_i,S_{i+1}))) \end{cases}$$

2.3. Modification des bornes de la quantité disponible pour les actions de synchronisation

1ère propagation

En diminuant B_{sup} ou en augmentant B_{inf} concernant la disponibilité de la ressource r pour une action de synchronisation (l'action S_i par exemple), on réduit la taille de l'intervalle contenant la disponibilité pour cette action ($Q(S_i)$).

Grâce aux contraintes:

$$\begin{cases} B_{inf}(Q(S_i)) \geq B_{inf}(Q(S_{i-1})) - B_{sup}(Q_{dec}(S_{i-1},S_i)) + B_{inf}(Q_{inc}(S_{i-1},S_i)) \\ B_{sup}(Q(S_i)) \leq B_{sup}(Q(Si\text{-}_1)) - B_{inf}(Q_{dec}(S_{i-1},S_i)) + B_{sup}(Q_{inc}(S_{i-1},S_i)) \\ B_{inf}(Q(S_i)) \geq B_{inf}(Q(S_{i+1})) + B_{inf}(Q_{dec}(S_i,S_{i+1})) - B_{sup}(Q_{inc}(S_i,S_{i+1})) \\ B_{sup}(Q(S_i)) \leq B_{sup}(Q(S_{i+1})) + B_{sup}(Q_{dec}(S_i,S_{i+1})) - B_{inf}(Q_{inc}(S_i,S_{i+1})) \end{cases}$$

la réduction de l'intervalle de $Q(S_i)$ est propagée pour les actions de synchronisation voisines S_{i-1} et S_{i+1}.

2ème propagation

La réduction de l'intervalle de disponibilité pour l'action S_i va aussi agir sur le nombre maximal d'instances possibles pour les actions multi-instances possiblement exécutées entre S_i et S_{i+1} grâce aux contraintes :

$$\begin{cases} nb(\alpha) \leq ((B_{sup}(S_i) - q_{inf}(r, \alpha) - \sum_{\beta \in A'ic(r)} q_{dec}(r, \beta))/q_{dec}(r, \alpha)) + 1 \\ nb(\alpha) \leq ((B_{inf}(S_i) - q_{sup}(r, \alpha) - \sum_{\beta \in A'ig(r)} q_{inc}(r, \beta))/q_{inc}(r, \alpha)) + 1 \end{cases}$$

2.4. Exclusion d'un lien *Prod* ou *Mod*

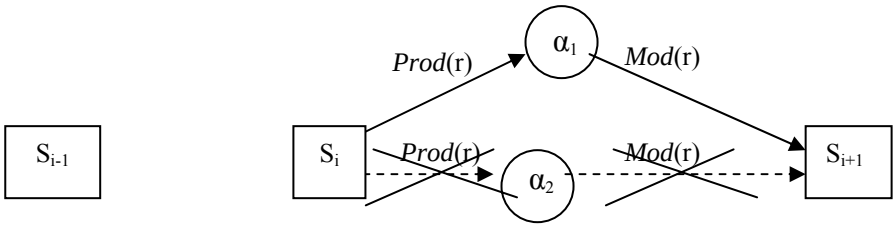

Figure 5.11: Exclusion d'un lien *Prod* ou *Mod*

L'exclusion d'un lien *Prod* ou *Mod* d'une action α engendre l'exclusion de cette action pour l'intervalle de temps limité par les deux actions de synchronisation reliées par ces liens (par exemple S_i et S_{i+1}).

Cette exclusion provoque un changement dans $B_{sup}(Q_{inc}(S_i,S_{i+1}))$ et/ou de $B_{sup}(Q_{dec}(S_i,S_{i+1}))$ selon le type de l'action à exclure (une action d'incrémentation ou de décrémentation ou hybride).

Les contraintes :

$$\begin{cases} B_{inf}(Q(S_i)) \geq B_{inf}(Q(S_{i+1})) + B_{inf}(Q_{dec}(S_i,S_{i+1})) - B_{sup}(Q_{inc}(S_i,S_{i+1})) \\ B_{sup}(Q(S_i)) \leq B_{sup}(Q(S_{i+1})) + B_{sup}(Q_{dec}(S_i,S_{i+1})) - B_{inf}(Q_{inc}(S_i,S_{i+1})) \end{cases}$$

doivent être respectées, ce qui peut provoquer un changement éventuel dans les bornes de $Q(S_i)$ et/ou $Q(S_{i+1})$.

2.5. Modification du nombre d'instances possibles d'une action

Pour deux actions de synchronisation successives S_{i-1} et S_i , et pour chaque action multi-instances α possiblement exécutée dans l'intervalle de temps limité par ces deux actions de synchronisation, la modification du nombre d'instances possibles de cette action peut provoquer un changement dans les bornes de S_{i-1} et/ou S_i dû au changement de $B_{sup}(Q_{inc}(S_i,S_{i+1}))$ et/ou de $B_{sup}(Q_{dec}(S_i,S_{i+1}))$ selon le type de l'action α (une action d'incrémentation ou de décrémentation ou hybride) et aux contraintes :

$$\begin{cases} B_{inf}(Q(S_i)) \geq B_{inf}(Q(S_{i+1})) + B_{inf}(Q_{dec}(S_i,S_{i+1})) - B_{sup}(Q_{inc}(S_i,S_{i+1})) \\ B_{sup}(Q(S_i)) \leq B_{sup}(Q(S_{i+1})) + B_{sup}(Q_{dec}(S_i,S_{i+1})) - B_{inf}(Q_{inc}(S_i,S_{i+1})) \end{cases}$$

Si le nombre maximal d'instances possibles devient égal à 0, l'action α est exclue de cet intervalle de temps.

3. AJOUT D'UNE ACTION DANS LE PLAN PARTIEL : MECANISME DE BRANCHEMENT

Tout plan contient au minimum deux actions de synchronisation, l'une juste après l'action *START* et l'autre juste avant l'action *END*.

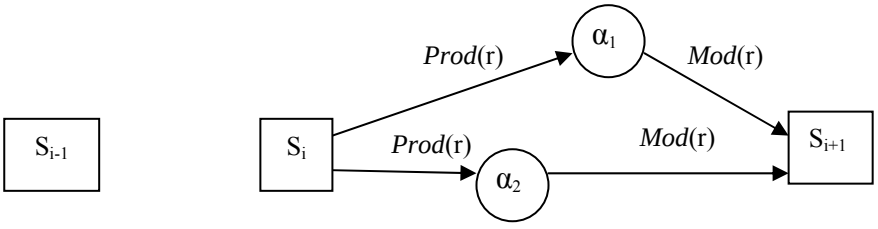

Figure 5.12: Mécanisme de branchement

Toute action insérée dans le plan doit être limitée par deux actions de synchronisation successives S_i et S_{i+1}: S_i la précède et S_{i+1} la suit.

Donc, pour ajouter une action α dans le plan partiel, on doit choisir entre deux possibilités :

- la première possibilité est d'insérer une nouvelle action de synchronisation S_i qui précède l'action α.
- la deuxième possibilité est de chercher les actions de synchronisation qui peuvent être exécutées dans l'intervalle de temps de α et essayer d'insérer cette dernière entre les deux actions de synchronisation successives trouvées tout en respectant les contraintes liées aux ressources.

3.1. Première possibilité: insertion d'une nouvelle action de synchronisation

Il est possible d'introduire, pour chaque nouvelle action α insérée dans le plan partiel, une action de synchronisation. Le problème qui pourrait être rencontré est que cela risque de générer une explosion combinatoire avec l'augmentation du nombre des actions du problème initial. En plus, deux actions qui ne sont pas dans le même intervalle de temps limité par deux actions de synchronisation successives ne peuvent pas s'exécuter en parallèle.

Cependant, on peut parfois être obligé d'ajouter une nouvelle action de synchronisation même s'il en existe déjà dans le plan partiel certaines qui pourraient précéder l'action α.

En effet, deux actions en mutex ne peuvent pas s'exécuter en parallèle. C'est pourquoi on doit les mettre dans deux intervalles différents si elles influent sur la même ressource.

Plus généralement, en introduisant une nouvelle action α dans le plan partiel, on doit vérifier, pour chaque ressource r qu'elle manipule (par consommation ou par production) et dans chaque intervalle délimité par des actions de synchronisation existantes relatives à cette ressource r, s'il existe une action β appartenant déjà au plan partiel qui est en mutex avec α. Si c'est le cas, alors α ne peut pas être dans le même intervalle de temps que β.

Si α ne peut appartenir à aucun intervalle de temps existant, on doit insérer une nouvelle action de synchronisation selon la position de l'action α par rapport aux autres action qui sont en exclusion mutuelle avec α.

Remarque 5.6

Les actions qui assignent la disponibilité d'une ressource r à une valeur fixe sont en mutex avec toutes les actions manipulant cette ressource par génération ou par consommation. Pour cette raison, on doit introduire, pour chaque action d'assignation α insérée dans le plan partiel, une action de synchronisation S_i qui précède α (sauf si α est la première action « concrète » à être insérée dans le plan partiel) et aucune autre action manipulant la ressource r ne peut appartenir à l'intervalle de temps limité par les deux actions de synchronisation successives S_i et S_{i+1}

3.2. Deuxième choix : utilisation des actions de synchronisation existantes

Ce deuxième choix consiste à suivre les étapes suivantes :

- chercher les actions de synchronisation qui peuvent être exécutées dans l'intervalle de temps de l'action α.

- pour chaque sous-ensemble composé de deux actions de synchronisation successives trouvées (S_i et S_{i+1}), vérifier si l'action α peut être insérée en respectant les contraintes :

$$\begin{cases} B_{inf}(Q(S_i)) \geq q_{inf}(r, \alpha) + \sum_{\beta \in A'ic(r)/\{\alpha\}} (q_{dec}(r, \beta)) \\ B_{sup}(Q(S_i)) \leq q_{sup}(r, \alpha) - \sum_{\beta \in A'ig(r)/\{\alpha\}} (q_{inc}(r, \beta)) \end{cases}$$

Si c'est le cas, alors on insère une instance de α comme action possiblement exécutée entre S_i et S_{i+1}.

Précédemment, nous avons déduit que la contrainte suivante :

$B_{sup}(Q(S_i)) - B_{inf}(Q_{dec}(S_i,S_{i+1})) + B_{inf}(Q_{inc}(S_i,S_{i+1}) \leq B_{sup}(Q(S_{i+1}))$

doit être respectée.

Cette contrainte met en relation les actions réellement introduites dans le plan partiel et les actions possiblement introduites dans le plan partiel.

En plus, un plan valide doit respecter la contrainte suivante :

$B_{sup}(Q(S_i)) - B_{inf}(Q_{dec}(S_i,S_{i+1})) + B_{inf}(Q_{inc}(S_i,S_{i+1}) \leq B_{sup}(Q(S_{i+1}))$.

Cette contrainte, qui ne traite que les actions réellement introduites dans le plan partiel, est plus restrictive que la contrainte précédente. Elle indique que, pour l'action de synchronisation S_{i+1}, la borne inférieure de la disponibilité de la ressource r ne doit pas dépasser la borne inférieure de la disponibilité de l'action de synchronisation qui la précède, c'est-à-dire S_i à laquelle on ajoute les quantités de r réellement produites et on déduit les quantités de r réellement consommées.

Elle garantit ainsi la validité du plan partiel au moment de l'activation des liens pour une action donnée entre deux actions de synchronisation tout en restant indépendant des actions candidates qui ne sont pas encore introduites dans le plan partiel.

Après implémentation de ces règles dans le mécanisme de branchement du planificateur CPT, celui-ci devient capable de traiter les problèmes de planification temporelle avec ressources selon la méthode basée sur les actions de synchronisation.

CONCLUSION

La méthode basée sur les actions de synchronisation permet d'améliorer la qualité des plans trouvés par rapport à celle basée sur les liens de ressource dans plusieurs problèmes de planification temporelle contenant des ressources rechargeables par incrémentation. En plus, cette méthode a l'avantage de gérer les actions hybrides et les actions de consommation par assignation qui ne peuvent pas être gérées par la méthode basée sur les liens de ressources.

Cependant, pour les problèmes où toutes les recharges se font par assignation, la méthode basée sur les actions de synchronisation se révèle inefficace car le plan trouvé par cette méthode est le même que celui trouvé par la méthode basée sur les liens de ressources puisque, dans ce cas, toutes les actions génératrices d'une ressource sont en exclusion mutuelle avec toutes les actions consommatrices ou génératrices de cette même ressource.

CONCLUSION ET PERSPECTIVES

Dans ce travail, nous avons amélioré l'expressivité du planificateur temporel parallèle optimal CPT en lui ajoutant la capacité de gérer des ressources sous une multitude de formes grâce à deux méthodes. La première est basée sur la notion de lien de ressources et la deuxième est basée sur celle d'action de synchronisation. Le planificateur obtenu préserve toute la puissance initiale du planificateur CPT pour le traitement des problèmes de planification temporelle. Le critère d'optimisation considéré reste toujours la durée totale des plans trouvés.

La méthode basée sur les liens de ressources se révèle intéressante pour les problèmes de planification temporelle intégrant des ressources sous des formes simples et limitant leur production aux actions d'assignation. Elle donne des résultats satisfaisants en un temps de recherche raisonnable.

La méthode basée sur les actions de synchronisation vise à améliorer à la qualité des plans obtenus en autorisant des cas de parallélisme entre des actions consommant et/ou produisant la même ressource qui sont interdits par la méthode basée sur les liens de ressources. La phase de prétraitement, réalisée par propagation des contraintes de ressources, est primordiale pour cette méthode car l'ajout d'actions de synchronisation dans un problème de planification augmente considérablement l'espace de recherche correspondant.

D'après les études expérimentales que nous avons effectuées, nous constatons que la méthode basée les liens de ressources peut résoudre un grand nombre de problèmes de planification temporelle avec ressources. En plus, elle permet de conserver les performances du planificateur CPT pour résoudre les problèmes de planification temporelle sans ressources.

L'intérêt de la méthode basée sur les actions de synchronisation se révèle particulièrement à prendre en considération d'autres possibilités de parallélisme des actions. En plus elle prend en charge les actions appelées actions hybrides qui permettent de consommer et de générer la même ressource en même temps. Elles sont cependant inefficaces lorsqu'il s'agit de problèmes où la seule possibilité de générer les ressources est par assignation de leur disponibilité à une valeur fixe car, dans ce cas, le plan trouvé par la méthode basée sur les actions de synchronisation est le même que celui trouvé par la méthode basée sur les liens de ressources. Une future implémentation de la solution basée sur les actions de synchronisation permettra d'analyser d'une manière plus concrète l'apport de cette méthode au niveau du temps de résolution des problème de planification et au niveau de la qualité des plans trouvés.

Il existe d'autres formes d'interaction entre le temps, les ressources et les actions que nous n'avons pas prises en considération dans notre travail car le formalisme que nous avons élaboré pour l'expression des ressources doit rester compatible avec le formalisme de Smith et Weld [Smith et Weld 1999] utilisé par le planificateur CPT pour exprimer les données temporelles. Ce formalisme est moins expressif que PDDL2.1 [Fox et Long 2001] qui est considéré comme un standard pour la planification temporelle,. Néanmoins la majorité des planificateurs temporels actuels sont incapables d'utiliser toute la puissance expressive de ce formalisme.

Dans le troisième niveau d'expressivité du formalisme PDDL2.1, la description des effets temporels et des conditions temporelles est plus développée que celle proposée dans le formalisme de Smith et Weld notamment dans la typologie des effets et des conditions qui peuvent être de début, de fin ou invariants. La prise en compte de cette typologie pour la représentation des données temporelles dans le planificateur CPT peut engendrer d'autres formes de représentation des ressources et des actions qui les manipulent, ce qui pourrait nécessiter des modifications même dans les méthodes de gestion des ressources élaborées.

L'utilisation de toute la puissance expressive du langage PDDL2.1 permetterait aussi d'améliorer la gestion des ressources dans le planificateur CPT avec la possibilité de spécifier les critères d'optimisation des plans valides obtenus qui peuvent être exprimés en fonction du temps et/ou des ressources consommées. Pour réaliser une telle amélioration, nous devons élaborer d'autres fonctions heuristiques pour le choix des actions candidates dans la phase de

branchement. Ces fonctions qui prendront en considération les nouveaux critères d'optimisation doivent permettre de préserver la puissance du planificateur dans le cas où le critère d'optimisation est la durée d'exécution du plan trouvé et elles doivent ordonner les actions candidates dans une opération de branchement d'une manière qui permette d'obtenir le plan valide et optimal en minimisant les opérations de backtracking.

Dans le quatrième niveau d'expressivité de PDDL2.1, il est possible d'exprimer des actions d'assignation dont la durée varie selon la quantité de ressources qu'elles génèrent ou consomment. La prise en compte de ce type d'action dans le planificateur CPT nécessiterait des modifications importantes dans la phase de prétraitement et la phase de branchement.

Il serait envisageable de prendre en considération les travaux réalisés dans d'autres classes de planification comme la planification dans l'incertain pour que les problèmes de planification traités et les plans obtenus avec le planificateur CPT soient plus proches de ceux attendus. Une incertitude peut être exprimée dans la définition des résultats relatifs à l'exécution des actions non déterministes ou encore dans la description des états déduits ou même observés.

La prise en considération de l'incertitude dans les actions non déterministes par exemple nécessiterait l'application des travaux réalisés dans ce domaine pour comparer les états calculés et les états observés et pour effectuer des modifications et des corrections sur le plan trouvé selon l'observation constatée.

ANNEXES

ANNEXE A : CODAGE D'UNE INSTANCE DU PROBLEME ZENOTRAVEL SELON LE FORMALISME STRIPS

Prédicats

Person(x)

Aircraft(x)

City(x)

Fuel_level(x)

At(x, c)

In(p, a)

Level(a, f)

Next(f_1, f_2)

Etat initial

{Person(P_1), Person(P_2), Aircraft(A_1), City(V_1), City(V_2), Fuel_level(F_0), Fuel_level(F_1), Fuel_level(F_2), Fuel_level(F_3), Fuel_level(F_4), At(P_1, V_2), At(P_2, V_2), At(A_1, V_1), Next(F_0, F_1), Next(F_1,F_2), Next(F_2,F_3), Next(F_3,F_4)}

Actions

Board(A_i, P_j, V_k)

Prec ={Aircraft(A_i), Person(P_j), City(V_k), *At*(A_i, V_k),*At*(P_j, V_k)}

Add = {*In*(P_j, A_i)}

Del = {*At*(P_j, V_k)}

Debark(A_i, P_j, V_k)

Prec ={Aircraft(A_i), Person(P_j), City(V_k), *At*(A_i, V_k), *In*(P_j, A_i)}

Add = {*At*(P_j, V_k)}

Del = {*In*(A_i, P_j)}

Fly(A_i, V_j, V_k, F_m, F_n)

Prec = {Aircraft(A_i), City(V_j), City(V_k), Fuel_level(F_m), Fuel_level(F_n), At(A_i,V_j), Level(A_i,F_m), Next(F_n,F_m)}

Add = { At(A_i, V_k), Level(A_i,F_n)}

Del = { At(A_i, V_j), Level(A_i,F_m)}

Zoom(A_i, V_j, V_k, F_l, F_m, F_n)

Prec = {Aircraft(A_i), City(V_j), City(V_k), Fuel_level(F_l), Fuel_level(F_m), Fuel_level(F_n), At(A_i,V_j), Level(A_i,F_l), Next(F_m,F_l), Next(F_n,F_m)}

Add = { At(A_i, V_k), Level(A_i,F_n)}

Del = { At(A_i, V_j), Level(A_i,F_l)}

Refuel(A_i, V_j, F_m, F_n)

Prec = {Aircraft(A_i), City(V_j), Fuel_level(F_m), Fuel_level(F_n), At(V_j, A_i), Level(A_i, F_m), Next(F_m, F_n)}

Add = {Level(A_i, F_n)}

Del = {Level(A_i, F_m)}

Etat but

{At(V_1, P_1), At(V_1, P_2), At(V_1, A)}

ANNEXE B : CODAGE DU PROBLEME ZENOTRAVEL SELON LE FORMALISME PDDL

Annexe B-1 : codage paramétré du problème ZenoTravel selon PDDL

```
(define (domain zeno_travel)

(:requirements :typing)

(:types aircraft person city flevel - object)

(:predicates
        (at ?x - (either person aircraft) ?c - city)
        (in ?p - person ?a - aircraft)
        (fuel-level ?a - aircraft ?l - flevel)
        (next ?l1 ?l2 - flevel))

(:action board
        :parameters (?p - person ?a - aircraft ?c - city)
        :precondition (and (at ?p ?c)(at ?a ?c))
        :effect (and (not (at ?p ?c))(in ?p ?a)))

(:action debark
        :parameters (?p - person ?a - aircraft ?c - city)
        :precondition (and (in ?p ?a)(at ?a ?c))
       :effect (and (not (in ?p ?a))(at ?p ?c)))

(:action fly
        :parameters (?a - aircraft ?c1 ?c2 - city ?l1 ?l2 - flevel)
```

```
        :precondition (and (at ?a ?c1)(fuel-level ?a ?l1)(next ?l2 ?l1))
        :effect (and (not (at ?a ?c1))(at ?a ?c2)
                        (not (fuel-level ?a ?l1))(fuel-level ?a ?l2)))

(:action refuel
        :parameters (?a - aircraft ?c - city ?l - flevel ?l1 - flevel)
        :precondition (and (fuel-level ?a ?l)(next ?l ?l1)(at ?a ?c))
        :effect (and (fuel-level ?a ?l1) (not (fuel-level ?a ?l)))))
```

Annexe B-2 : codage d'une instance du problème ZenoTravel selon PDDL

```
 (define (problem ZTRAVEL-1-2)

(:domain zeno_travel)

(:objects
plane1 - aircraft
person1 - person
        person2 - person
        city1 - city
        city2 - city
        fl0 - flevel
        fl1 - flevel
        fl2 - flevel
        fl3 - flevel
        fl4 – flevel)

(:init
        (at plane1 city1)
```

```
        (fuel-level plane1 fl1)
        (at person1 city2)
        (at person2 city2)
        (next fl0 fl1)
        (next fl1 fl2)
        (next fl2 fl3)
        (next fl3 fl4))

(:goal (and(at plane1 city1)(at person1 city1)(at person2 city1))))
```

ANNEXE C : CODAGE DU PROBLEME ZENOTRAVEL DANS LA FORME NUMERIQUE SELON LE FORMALISME PDDL 2.1

Annexe C-1 : définition des paramètres du problème ZenoTravel dans la forme numérique selon PDDL 2.1

```
(define (domain zeno-travel)

(:requirements :durative-actions :typing :fluents)

(:types aircraft person city - object)

(:predicates    (at ?x - (either person aircraft) ?c - city)
                (in ?p - person ?a - aircraft)
)

(:functions     (fuel ?a - aircraft)
        (distance ?c1 - city ?c2 - city)
        (slow-speed ?a - aircraft)
        (fast-speed ?a - aircraft)
         (capacity ?a - aircraft)
        (refuel-rate ?a - aircraft)
        (total-fuel-used)
        (boarding-time)
        (debarking-time)
)

(:durative-action board
        :parameters (?p - person
                        ?a - aircraft
```

```
                    ?c - city)
        :duration (= ?duration (boarding-time))
        :condition (and (at start (at ?p ?c))(over all (at ?a ?c)))
         :effect (and (at start (not (at ?p ?c))) (at end (in ?p ?a)))
)

(:durative-action debark
        :parameters (?p - person
                    ?a - aircraft
                    ?c - city)
        :duration (= ?duration (debarking-time))
        :condition (and (at start (in ?p ?a))(over all (at ?a ?c)))
        :effect (and (at start (not (in ?p ?a)))(at end (at ?p ?c)))
)

(:durative-action fly
        :parameters (?a - aircraft
                    ?c1 ?c2 - city)
        :duration (= ?duration (/ (distance ?c1 ?c2)(slow-speed ?a)))
        :condition (and (at start (at ?a ?c1))(at start (>= (fuel ?a)
             (* (distance ?c1 ?c2) (slow-burn ?a)))))
        :effect (and (at start (not (at ?a ?c1)))
             (at end (at ?a ?c2))
             (at end (increase total-fuel-used (* (distance ?c1 ?c2) (slow-burn ?a))))
             (at end (decrease (fuel ?a) (* (distance ?c1 ?c2) (slow-burn ?a)))))
)

(:durative-action zoom
        :parameters (?a - aircraft
                    ?c1 ?c2 - city)
        :duration (= ?duration (/ (distance ?c1 ?c2)(fast-speed ?a)))
        :condition (and (at start (at ?a ?c1)(at start (>= (fuel ?a)
                    (* (distance ?c1 ?c2) (fast-burn ?a)))))
```

```
        :effect (and (at start (not (at ?a ?c1)))
                (at end (at ?a ?c2))
                (at end (increase total-fuel-used (* (distance ?c1 ?c2) (fast-burn ?a))))
                (at end (decrease (fuel ?a) (* (distance ?c1 ?c2) (fast-burn ?a)))))
)

(:durative-action refuel
        :parameters (?a - aircraft ?c - city)
        :duration (= ?duration (/ (- (capacity ?a) (fuel ?a)) (refuel-rate ?a)))
       :condition (and (at start (> (capacity ?a) (fuel ?a)))(over all (at ?a ?c)))
       :effect (at end (assign (fuel ?a)(capacity ?a))))
)
```

Annexe C-2 : codage d'une instance du problème ZenoTravel dans la forme numérique selon PDDL 2.1

```
(define (problem ZTRAVEL-1-2)

(:domain zeno-travel)

(:objects
        plane1 - aircraft
        person1 - person
        person2 - person
        city0 - city
        city1 - city
        city2 - city
)

(:init
        (at plane1 city0)
        (= (slow-speed plane1) 198)
        (= (fast-speed plane1) 449)
```

```
        (= (capacity plane1) 10232)
        (= (fuel plane1) 3956)
        (= (refuel-rate plane1) 2904)
        (at person1 city0)
        (at person2 city2)
        (= (distance city0 city0) 0)
        (= (distance city0 city1) 678)
        (= (distance city0 city2) 775)
        (= (distance city1 city0) 678)
        (= (distance city1 city1) 0)
        (= (distance city1 city2) 810)
        (= (distance city2 city0) 775)
        (= (distance city2 city1) 810)
        (= (distance city2 city2) 0)
        (= (total-fuel-used) 0)
        (= (boarding-time) 0.3)
        (= (debarking-time) 0.6)
)

(:goal (and (at plane1 city1)(at person1 city2)(at person2 city0)))

(:metric minimize (+ (* 4 (total-time))  (* 0.005 (total-fuel-used)))))
```

ANNEXE D : CODAGE DU PROBLEME DRIVERLOG SELON LE FORMALISME PDDL 2.1 - DEFINITION DES PARAMETRES DU PROBLEME

```
(define (domain driverlog)

(:requirements :typing :durative-actions :fluents)

(:types         location locatable - object
                driver truck obj - locatable)

(:predicates    (at ?obj - locatable ?loc - location)
                (in ?obj1 - obj ?obj - truck)
                (link ?x ?y - location)
                (path ?x ?y - location)
                (empty ?v - truck)
)

(:functions     (time-to-walk ?loc ?loc1 - location)
                (time-to-drive ?loc ?loc1 - location)
                (fuel-consumption ?loc ?loc1 - location)
                (capacity-fuel ?v – truck)
                (capacity-obj ?v – truck)
                (fuel ?v – truck)
                (volume ?obj obj)
)

(:durative-action LOAD-TRUCK
:parameters     (?obj - obj
                ?truck - truck
                ?loc - location)
```

```
:duration      (= ?duration 2)
:condition     (and (over all (at ?truck ?loc))
               (at start (>=(capacity-obj ?truck)(+(volume ?obj))))
               (at start (at ?obj ?loc)))
:effect        (and (at start (not (at ?obj ?loc)))
               (at end (decrease(capacity-obj ?truck)(+(volume ?obj))))
               (at end (in ?obj ?truck)))
)

(:durative-action UNLOAD-TRUCK
:parameters    (?obj - obj
               ?truck – truck
               ?loc - location)
:duration      (= ?duration 2)
:condition     (and (over all (at ?truck ?loc))
               (at start (in ?obj ?truck)))
:effect        (and (at start (not (in ?obj ?truck)))
               (at end (increase(capacity-obj ?truck)(+(volume ?obj))))
               (at end (at ?obj ?loc)))
)

(:durative-action BOARD-TRUCK
:parameters    (?driver - driver
               ?truck - truck
               ?loc - location)
 :duration     (= ?duration 1)
 :condition    (and (over all (at ?truck ?loc))
               (at start (at ?driver ?loc))
               (at start (empty ?truck)))
:effect        (and (at start (not (at ?driver ?loc)))
               (at end (driving ?driver ?truck))
               (at start (not (empty ?truck))))
)
```

```
(:durative-action DISEMBARK-TRUCK
:parameters   (?driver - driver
               ?truck - truck
               ?loc - location)
:duration     (= ?duration 1)
:condition    (and (over all (at ?truck ?loc)
               (at start (driving ?driver ?truck)))
:effect       (and (at start (not (driving ?driver ?truck)))
               (at end (at ?driver ?loc))
                (at end (empty ?truck)))
)

(:durative-action DRIVE-TRUCK
 :parameters  (?truck - truck
               ?loc-from - location
               ?loc-to - location)
:duration     (= ?duration (time-to-drive ?loc-from ?loc-to))
:condition    (and (at start (at ?truck ?loc-from))
               (at start (link ?loc-from ?loc-to)))
               (at start (>= (fuel ?truck)(fuel-consumption ? loc-from ?loc-to))))
  :effect     (and (at start (not (at ?truck ?loc-from)))
               (at end (at ?truck ?loc-to))
               (at end (decrease (fuel ?truck)(+(fuel-consumption ? loc-from ?loc-to)))))
)

(:durative-action WALK
:parameters   (?driver - driver
               ?loc-from - location
               ?loc-to - location)
:duration     (= ?duration (time-to-walk ?loc-from ?loc-to))
:condition    (and (at start (at ?driver ?loc-from))
               (at start (path ?loc-from ?loc-to)))
```

```
:effect         (and (at start (not (at ?driver ?loc-from)))
                (at end (at ?driver ?loc-to)))
)

(:durative-action refuel
:parameters     (?truck - truck
                ?location – location)
:duration       (= ?duration 3)
:condition      (and (at start (> (capacity-fuel ?truck)(fuel ?truck)))
                (over all (at ?truck ?location))
:effect         (at end (assign (fuel ?truck)(capacity-fuel ?truck))))
)
```

TABLE DES FIGURES

LISTE DES TABLEAUX

BIBLIOGRAPHIE

[Aggoun et al. 1998]

A. Aggoun, N. Beldiceanu, E. Bourreau et H. Simonis : "L'apport des contraintes globales pour la modélisation et la résolution d'applications industrielles". In proceeding of FRANCORO II, Deuxièmes Journées Francophones de Recherche Opérationnelle, 1998.

[Anbulagan et Li 1997]

Anbulagan et C.M. Li: "Heuristics based on unit propagation for satisfiability problems". In Proceedings of IJCAI-97, pp. 366-371, 1997.

[Bayardo et Schrag 1998]

R. J. Bayardo Jr. et R. C. Schrag: "Using CSP Look-back techniques to solve real world SAT instances". In Proceedings of AAAI-98, pp. 203-208, 1998.

[Bessière et Régin 1997]

Bessière, C. et Régin, J.-C. : "Arc Consistency for General Constraint Networks: Preliminary Results". In Proceedings of IJCAI'97, 398-404.

[Blum et Furst 1995]

A. Blum et M. Furst: "Fast planning through Plan-graph analysis". In Proceedings of IJCAI-95, pp.1636-1642, 1995.

[Bonet et Geffner 1999]

B. Bonet et H. Geffner: "Planning As Heuristic Search: New results". In Proceedings of ECP-99, pp. 360-372, 1999.

[Bonet et Geffner 2001]

B. Bonet et H. Geffner: "Planning As Heuristic Search". Artificial Intelligence, 129(1-2), pp. 5-33, 2001.

[Brucker et al. 1999]

P. Brucker, A. Drexl, R. Moring, K. Neumann et E. Pesch: "Resource-constrained project scheduling: notation, classification, models and methods". European Journal of Operational Research ,112(1), pp. 3-41, 1999.

[Bylander 1991]

T. Bylander: "Complexity results for planning". In Proceedings of IJCAI-91, pp. 274-279, 1991

[Caseau et al. 1999]

Y. Caseau, F. X. Josset et F. Laburthe: "Claire: Combining sets, search and rules to better express algorithms". In Proceedings of ICLP-99, pp. 245-259, 1999.

[Caseau et Laburthe 1996]

Y. Caseau et F. Laburthe: "Introduction to the CLAIRE programming language". Rapport technique, Laboratoire en informatique de l'Ecole Normale Supérieure, 1996.

[Cayrol et al. 2000]

M. Cayrol, P. Régnier et V. Vidal: "New results about LCGP, a least committed Graphplan". In Proceedings of AIPS−2000, pp. 273-282, 2000.

[Cayrol et al. 2001]

M. Cayrol, P. Régnier et V. Vidal: "Least committed Graphplan". Artificial Intelligence, 130, pp. 85-118, 2001.

[Chen et Van Beek 1999]

X. Chen et P. Van Beek: "CPlan: A constraint programming approach to planning". In Proceedings of AAAI-99, pp. 585-590, 1999.

[Cohen et al. 1994]

B. Cohen, H. Kautz et B. Selman: "Noise strategies for local search". In Proceedings of AAAI-94, pp. 337-343, 1994.

[Fikes et Nikson 1971]

R.E. Fikes et N.J. Nilsson: "STRIPS: A new approach to the application of theorem proving to problem solving". Artificial Intelligence, 2, pp.189-208 , 1971.

[Fox et Long 1999]

M. Fox et D. Long: "The efficient implementation of the plan-graph in STAN". Journal of Artificial Intelligence Research, 10, pp. 87-115, 1999.

[Fox et Long 2001]

M. Fox et D. Long: "PDDL2.1: An extension to PDDL for expressing temporal planning domains". Technical report, Department of Computer Science, University of Durham, UK, 2001.

[Fox et Long 2002]

M. Fox et D. Long: "PDDL+: Planning with time and metric resources". Technical report, Department of Computer Science, University of Durham, UK, 2002.

[Freuder 1978]

E. Freuder: "Synthesizing constraint expressions". Communications of the ACM, 21, pp. 958-966, 1978.

[Garey et Johnson 1979]

M. R. Garey et D. S. Johnson: "Computers and intractability: A guide to the theory of NP-completeness". W.H. Freeman and Company, 1979.

[Geffner et Haslum 2000]

H. Geffner et P. Haslum: "Admissible heuristics for optimal planning". In Proceedings of AIPS-2000, pp. 140-149, 2000.

[Geffner et Haslum 2001]

H. Geffner et P. Haslum: "Heuristic planning with time and resources". In Proceedings of ECP-01, pp.121-132, 2001.

[Geffner et Vidal 2004]

H. Geffner et V. Vidal: "Branching and pruning: An optimal temporal POCL planner based on constraint programming". In Proceedings of AAAI-04, pp. 570-577, 2004.

[Geffner et Vidal 2005]

H. Geffner et V. Vidal: "Solving Simple Planning Problems with More Inference and No Search". In Proceedings of CP-2005, pp. 682-696, 2005.

[Gervini et Serina 2002]

A. Gerevini et I. Serina: "LPG: a Planner based on Local Search for Planning Graphs". In Proceedings of AIPS-02, pp. 13-22, 2002.

[Gervini et al. 2005]

A. Gerevini, A. Saetti, I. Serina et P. Toninelli: "Fast Planning in Domains with Derived Predicates: An Approach Based on Rule-Action Graphs and Local Search". In Proceedings of AAAI-05, pp. 1157-1162, 2005.

[Ghallab et al. 1988]

M. Ghallab, R. Alami et R. Chatila: "Dealing with time in planning and execution monitoring". In proceedings of the 4th International Symposium on Robotics Research, 1988.

[Ghallab et Laborie 1995]

M. Ghallab et P. Laborie: "Planning with sharable resources constraints". In Proceedings of IJCAI-95, pp. 1643-1649, 1995.

[Hoffman et Nebel 2001]

J. Hoffman et B. Nebel: "The FF planning system: Fast plan generation through heuristic search". Journal of Artificial Intelligence Research, 14, pp. 253-302, 2001.

[Kambhampati 2000]

S. Kambhampati: “Planning graph as a (dynamic) CSP: exploiting EBL, DDB and other CSP search techniques in GraphPlan”. Journal of Artificial Intelligence Research, 12, pp. 1-34, 2000.

[Kambhampati et Do 2000]

S. Kambhampati et M. B. Do: “Solving Planning Graph by Compiling it into a CSP”. In Proceeding of AIPS-2000, pp. 82-91, 2000.

[Kambhampati et Do 2001]

S. Kambhampati et M. B. Do: “Sapa: A Domain-Independent Heuristic Metric Temporal Planner”. In Proceedings of ECP-01, 2001.

[Kambhampati et Do 2003]

S. Kambhampati et M. B. Do: “Improving the Temporal Flexibility of Position Constrained Metric Temporal Plans”. In Proceedings of ICAPS-03, pp. 42-51, 2003.

[Kambhampati et Nguyen 2001]

S. Kambhampati et X.L. Nguyen: "Reviving Partial Order Planning". In Proceedings of IJCAI-2001, pp. 459-466, 2001.

[Kautz et al. 1996]

H. Kautz, D. McAllester et B. Selman: “Encoding plans in propositional logic”. In Proceedings of KR-96, pp. 374-384, 1996.

[Kautz et Selman 1992]

H. Kautz et B. Selman: “Planning As Satisfiability”. In Proceedings of ECAI-92, pp. 1194-1201,1992.

[Kautz et Selman 1999]

H. Kautz et B. Selman: “Unifying SAT-based and Graph-based Planning”. In Proceedings of IJCAI-99, pp. 318-325, 1999.

[Kautz 2004]

H. Kautz, “SatPlan04: Planning As Satisfiability”. In Abstracts of IPC4, 2004.

[Kautz et al. 2006]

H. Kautz, B. Selman et J. Hoffmann: “SatPlan06: Planning As Satisfiability”. In Abstracts of IPC5, 2006.

[Laborie 2003]

P. Laborie: “Algorithms for propagating resource constraints in AI planning and scheduling”. Artificial Intelligence, 143(2), pp. 151-188, 2003.

[Laburthe 2000]

F. Laburthe: “Choco: implementing a CP kernel”. Workshop on Techniques for Implementing Constraint programming Systems (TRICS-98), Lecture Notes in Computer Science, 1894, 2000.

[Lhomme 1993]

O. Lhomme: "Consistency techniques for numeric CSPs" In Proceedings of IJCAI-93, pp. 232-238, 1993.

[McAllester et Rosenblitt 1991]

D. McAllester et D. Rosenblitt: “Systematic nonlinear planning”. In Proceedings of AAAI-91, pp. 634-639, 1991.

[McDermott 1998]

D. V. McDermott: “PDDL: The Planning Domain Definition Language”. Technical Report, Yale University, 1998.

[McDermott 2000]

D. V. McDermott: “The 1998 AI planning systems competition”. Artificial Intelligence Magazine, 21(2), pp. 35-55, 2000.

[Mittal et Falkenhainer 1990]

S. Mittal et B. Falkenhainer : "Dynamic constraint satisfaction problems". In Proceedings of AAAI-90, pp. 25-32, 1990.

[Nebel et al. 1997]

B. Nebel , Y. Dimopoulos et J. Koehler "Ignoring irrelevant facts and operators in plan generation". In Proceedings of ECP−97, pp. 338-350, 1997.

[Pednault 1989]

E. P. D. Pednault: "ADL: Exploring the middle ground between STRIPS and the situation calculus". In Proceedings of KR−89, pp. 324-332, 1989.

[Penberthy et Weld 1992]

S. Penberthy et D. Weld: "UCPOP: A sound, complete partial order planner for ADL". In Proceedings of KR-92, pp. 103-114, 1992.

[Smith et Weld 1999]

D. E. Smith et D. S. Weld: "Temporal planning with mutual exclusion reasoning". In Proceedings of IJCAI-99, pp. 326-333, 1999.

[Waltz 1975]

D. L. Waltz: "Generating semantic descriptions from drawings of scenes with shadows". In The Psychology of Computer Vision McGraw Hill, pp. 19-91, 1975.

[Weld 1994]

D. S. Weld: "An introduction to Least Commitment Planning". Artificial Intelligence Magazine, 15, pp. 93-123, 1994.

[Xing et al. 2006]

Z. Xing, Y. Chen et W. Zhang: "MaxPlan: optimal planning by Decomposed Satisfiability and Backward Reduction". In proceedings of ICAPS-06, pp. 442-447, 2006.

[Younes et Simmons 2003]

Håkan L. S. Younes et Reid G. Simmons : "VHPOP: Versatile heuristic partial order planner". Journal of Artificial Intelligence Research, 20, pp. 405-430, 2003.

RESUME

Dans cette thèse, nous avons pour objectif d'intégrer la gestion des ressources consommables et renouvelables dans la planification temporelle en choisissant comme critère d'optimalité le temps d'exécution totale du plan trouvé.

D'un point de vue pratique, nous cherchons à améliorer le planificateur CPT, qui ne gère pas les ressources dans sa version initiale. Nous tenons aussi à conserver les performances initiales de ce planificateur pour résoudre les problèmes de planification temporelle.

Pour réaliser ce but nous avons établi un formalisme permettant d'exprimer les contraintes liées aux ressources dans le planificateur CPT. Ce formalisme est en relation étroite avec le formalisme utilisé par ce planificateur pour exprimer les contraintes temporelles.

Nous avons proposé deux méthodes différentes pour la gestion des ressources dans le planificateur CPT. La première, basée sur les liens de ressources, exige des contraintes additionnelles pour les ressources qui doivent être respectées. Elle permet de résoudre des problèmes de planification temporelle avec ressources dans des délais très intéressants.

La deuxième méthode est basée sur les actions de synchronisation. Elle est moins restrictive que la première méthode et les contraintes qu'elle exige sont moins sévères. Plusieurs règles de propagations des contraintes liées aux ressources ont été introduites avec cette méthode pour accélérer la recherche. Dans beaucoup de cas, cette méthode permet de calculer des plans valides optimaux d'une qualité meilleure que celle de la méthode basée sur les liens de ressources.

ABSTRACT

In this thesis, we aim at introducing the management of consumable and renewable resources in temporal, parallel and domain-independent planning.

As an application of our work, our purpose is to improve the temporal parallel planner named CPT which is not able to manage resources in its original version. In addition, we want to preserve all the original performances of this planner on temporal planning problems. We focus on the total duration of the plan as the optimization criterion.

To express resource constrains, we develop a resource-based formalism in relation of the mutual-exclusion-based one introduced by Smith and Weld and used by the CPT planner to express temporal constraints.

We propose two different methods to manage resources in the CPT planner: The first one, based on resource links, requires additional resource constraints which must be respected and can resolve temporal planning problems under resources within remarkable search delays. The second one is based on synchronization actions; it is less restrictive than the first method but it requires many additional propagation rules to speed up the search process.

In many cases, the synchronization actions-based method finds temporal optimal parallel plans with better quality than the ones found by the resource links-based method.