Planification de missions pour des systèmes robotiques maritimes intégrés

Nicolas Cavrel

THÈSE

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ GRENOBLE ALPES

École doctorale : MSTII - Mathématiques, Sciences et technologies de l'information, Informatique
Spécialité : Mathématiques et Informatique
Unité de recherche : Laboratoire d'Informatique de Grenoble

Planification de missions pour des systèmes robotiques maritimes intégrés

Mission planning for integrated marine robotic systems

Présentée par :

Nicolas CAVREL

Direction de thèse :

Damien PELLIER MAITRE DE CONFERENCES HDR, UNIVERSITE GRENOBLE ALPES	Directeur de thèse
Humbert FIORINO MAITRE DE CONFERENCES, UNIVERSITE GRENOBLE ALPES	Co-encadrant de thèse

Rapporteurs :

CHARLES LESIRE
DIRECTEUR DE RECHERCHE, ONERA
CAROLINE CHANEL
ENSEIGNANTE-CHERCHEUSE HDR, ISAE - SUPAERO

Thèse soutenue publiquement le **22 mai 2024**, devant le jury composé de :

DOMINIQUE VAUFREYDAZ, PROFESSEUR DES UNIVERSITES, UNIVERSITE GRENOBLE ALPES	Président
CHARLES LESIRE, DIRECTEUR DE RECHERCHE, ONERA	Rapporteur
CAROLINE CHANEL, ENSEIGNANTE-CHERCHEUSE HDR, ISAE - SUPAERO	Rapporteure
NADIA BRAUNER, PROFESSEURE DES UNIVERSITES, UNIVERSITE GRENOBLE ALPES	Examinatrice

Invités :

HUMBERT FIORINO
MAITRE DE CONFERENCES, UNIVERSITE GRENOBLE ALPES
DAMIEN PELLIER
MAITRE DE CONFERENCES HDR, UNIVERSITE GRENOBLE ALPES

Université Grenoble Alpes

Laboratoire d'Informatique de Grenoble
EXAIL Robotics

Planification de missions pour des systèmes robotiques maritimes intégrés

Nicolas Cavrel

1. Rapporteur	Charles Lesire
	ONERA
2. Rapporteure	Caroline Chanel
	ISAE-SUPAERO
Direction de thèse	Damien Pellier et Humbert Fiorino

Nicolas Cavrel
Planification de missions pour des systèmes robotiques maritimes intégrés
Rapporteurs : Charles Lesire et Caroline Chanel
Direction de thèse : Damien Pellier et Humbert Fiorino

Université Grenoble Alpes
Laboratoire d'Informatique de Grenoble
700 avenue Centrale Domaine Universitaire
38401 Saint Martin d'Hères

EXAIL Robotics
262 rue des Frères Lumière
83130 La Garde

Résumé

Ces dernières années, l'utilisation de systèmes multi-robots dans des contextes opérationnels s'est fortement intensifiée. Notamment dans des environnements hostiles, ces systèmes permettent d'opérer avec efficacité et sécurité. Les missions de déminage sous-marin sont des exemples typiques d'opérations où des appareils autonomes sont particulièrement intéressants. En effet, ces missions présentent à la fois des contraintes temporelles et environnementales fortes, tout en nécessitant l'emploi d'une flotte d'appareils hétérogènes. Ainsi, la production d'un plan d'action pour chacun de ces appareils se révèle être un problème central, mais également complexe.

La planification automatique est la branche de l'informatique qui cherche à apporter une solution automatique à ce problème. L'environnement de la mission est modélisé au moyen de propriétés dont les valeurs sont modifiées par les actions des appareils de la flotte. À partir de cette modélisation, les systèmes de planification automatique construisent un enchaînement d'actions permettant d'atteindre un objectif défini a priori. La planification hiérarchique est un type particulier de planification dans lequel l'objectif de la mission est défini par un ensemble de tâches à réaliser.

L'objectif des travaux présentés ci-dessous est, dans un premier temps, de passer en revue les approches de planification automatique existantes, d'abord dans le contexte du déminage sous-marin, puis dans un contexte plus général. Dans un deuxième temps, nous proposons de nouvelles approches permettant de résoudre informatiquement des problèmes de planification hiérarchique. Nous présentons théoriquement ces approches, puis démontrons expérimentalement leur efficacité par rapport aux approches de la littérature.

Abstract

In recent years, the use of multi-robot systems in operational contexts has significantly increased. Particularly in hostile environments, these systems allow for efficient and secure operations. Underwater demining missions are typical examples of operations where autonomous devices are particularly valuable. Indeed, these missions involve both strong temporal and environmental constraints, while requiring the use of a fleet of heterogeneous devices. Thus, devising an action plan for each of these devices proves to be a central but also complex problem.

Automated planning is the branch of computer science that aims at providing an automatic solution to this problem. The mission environment is modeled using properties whose values are modified by the actions of the devices. From this modeling, automated planning systems construct a sequence of actions to achieve a predefined objective. Hierarchical planning is a specific type of planning in which the objective of the mission is defined by a set of tasks to be accomplished.

The aim of the work presented below is, first and foremost, to review existing automated planning approaches, initially in the context of underwater demining and then in a more general context. Secondly, we propose new approaches to solve hierarchical planning problems. We theoretically present these approaches and then experimentally demonstrate their effectiveness compared to approaches in the literature.

Remerciements

Je tiens tout d'abord à remercier mes deux directeurs de thèse, Damien et Humbert, sans qui rien de tout cela n'aurait été possible. Merci pour votre confiance, pour vos conseils, et pour toutes ces discussions passées à discourir des liens de causalité. Merci de m'avoir guidé depuis le début, et de m'avoir permis d'évoluer en la personne que je suis.

À Naquima, pour tous ces entretiens plus ou moins scientifiques. Il est particulièrement étonnant de constater combien la science est prolifique quand elle est discutée dans l'amitié.

À ma famille, mes parents, mes frères et sœurs, qui ne se rendent certainement pas compte de leur importance dans ces travaux. En particulier à ma mère, assidue relectrice et commentatrice de ces lignes. À mon père, pour tout son soutien, et Laura, et Alice, qui surent m'inspirer chacune à leur manière, et qui savent beaucoup.

À tous mes amis, à ceux qui savent la valeur du Pique, du Cœur, du Trèfle et du Carro, à ceux qui savent l'importance de l'équipe de nuit.

À Margot, qui sait que si le temps est cruel, certaines choses lui résisteront toujours.

À Paul et Yvonne, que j'admire tant. Je me revois dévorer, dérober, ces viandes et ces livres, bien trop lourds pour moi, car l'un nourrit le corps et l'autre nourrit l'esprit. Nous sommes des nains sur des épaules de géants.

Table des matières

I Introduction et état de l'art **1**

1 Introduction **3**

2 Contexte général d'une mission de déminage **7**
2.1 Différents types de mines 7
2.2 Lutter contre les mines marines 9
2.2.1 Opérations de dragage des mines 10
2.2.2 Opérations de chasse aux mines 11
2.3 Déroulement d'une mission de chasse aux mines 12
2.3.1 Définition de la zone de mission 12
2.3.2 Phase de détection 14
2.3.3 Phase d'identification 17
2.3.4 Phase de neutralisation 18
2.3.5 Flottes intégrées MCM (Mines Counter Measures) 19
2.4 Définition du problème et Ambitions 21
2.4.1 Hypothèses et problème 21
2.4.2 Ambitions 23

3 État de l'art **25**
3.1 Planification de mission pour une flotte maritime intégrée 25
3.1.1 Approches par reformulation 27
3.1.2 Approches par planification automatique 30
3.1.3 Commentaire intermédiaire 35
3.2 Planification automatique : Un formalisme de planification général . 36
3.2.1 Décrire un problème de planification 38
3.2.2 Exemple de problème de planification 44
3.3 Classification des planificateurs existants 54
3.3.1 Classes de problèmes et de solutions 54
3.3.2 Catégorie 1 : Problèmes intrinsèquement séquentiels 62
3.3.3 Catégorie 2 : Problèmes mixtes 63
3.3.4 Catégories 3 : Problèmes intrinsèquement concurrents 68

3.3.5 Conclusions de l'état de l'art 69

II Contributions **71**

4 CTHD : Un encodage HTN vers STRIPS concurrent **75**
4.1 Définition du problème 76
4.2 CPFD : Un algorithme hiérarchique concurrent 79
4.3 CTHD : Un encodage HTN vers STRIPS concurrent 83
4.3.1 Modélisation du réseau de tâches et de la couche courante . . 84
4.3.2 Encodage de la procédure CPFD en actions STRIPS 84
4.4 Expérimentations 91
4.4.1 Configuration expérimentale 91
4.4.2 Commentaires 92
4.4.3 Discussion sur l'optimalité du makespan dans CTHD 94
4.5 Conclusion 94

5 TEP : Un planificateur hiérarchique et temporel **97**
5.1 Définition du problème 98
5.2 TEP : Temporal Event Planning 102
5.2.1 Représentation des réseaux de tâches temporelles 102
5.2.2 Principe de résolution 106
5.3 Expérimentations 115
5.3.1 Configuration expérimentale 116
5.3.2 Benchmarks de planification 116
5.3.3 Résultats 117
5.4 Conclusion 119

6 TTC : Temporal Task Converter **121**
6.1 Définition du problème 122
6.2 Temporal Task Converter : une compilation non temporelle 124
6.2.1 Compilation des tâches abstraites 125
6.2.2 Compilation des actions duratives 125
6.2.3 Compilation des réseaux de tâches temporelles 128
6.3 Expérimentations 129
6.3.1 Configuration expérimentale 129
6.3.2 Benchmarks de planification 130
6.3.3 Résultats 130
6.4 Conclusion 132

7 Conclusions et perspectives **133**

7.1 Bilan des contributions 133
7.2 Limites et perspectives d'évolution 135

Bibliographie **137**

Table des figures

2.1 Croquis d'une mine utilisée lors de la guerre de Sécession américaine en 1861. 9
2.2 Mine allemande datant de la Seconde Guerre mondiale. 9
2.3 Les différents types de mines en fonction de leur position dans l'eau. Les types 1 à 3 sont des mines de surface, quand les mines 4 à 7 sont des mines de fond. 9
2.4 Hélicoptère tractant une drague. 11
2.5 Un exemple de zone de mission, la zone est divisée en sous-zones, les caractéristiques environnementales sont supposées constantes sur chaque sous-zone. 13
2.6 Un A18 de la société Exail Robotics 14
2.7 Rails à suivre par l'appareil sous-marin pour cartographier la zone. . . 15
2.8 Un exemple de rails double, permettant deux prises de vue de chaque point. 15
2.9 Deux images SAS d'épaves dans la baie de Hyères. 16
2.10 Les étapes d'une phase d'identification classique. 17
2.11 Un appareil SeaScan d'Exail Robotics fournissant un contact visuel avec le MILCO. 18
2.12 Un Poisson Auto Propulsé (PAP), appareil autonome de neutralisation de mines. 19
2.13 Un drone KSter de Exail Robotics. 19
2.14 Vaisseau-mère M940 (Naval Group) 20
2.15 Appareil de surface autonome de support Inspector 125. 20
2.16 Vue récapitulative du système recherché dans le cadre de cette thèse. . 24

3.1 Représentation des différentes catégories de systèmes dans le contexte de la chasse aux mines. 26
3.2 Application d'une action a à un état $S1$. a est applicable à $S1$ car $pre(a) \subseteq S1$ et $S2 = \gamma(S1, a) = (S1 - del(a)) \cup add(a)$ 39

3.3 Représentation d'un plan solution, un plan solution est valide si les actions le composant sont successivement applicables aux états résultant de l'état initial et si l'état final contient l'état objectif G du problème. En d'autres termes si $\gamma(I, \langle a_1, ..., a_2 \rangle) \supseteq G$. 39
3.4 Décomposition d'un réseau de tâches au moyen d'une méthode. La tâche décomposée est remplacée par le réseau de tâches de la méthode. 42
3.5 Exemple de plan solution d'un problème hiérarchique, la suite d'actions doit résoudre un réseau de tâches issu de la décomposition du réseau initial. 43
3.6 Une solution d'un problème hybride doit répondre au problème hiérarchique tout en atteignant l'état objectif du problème. 43
3.7 Situation initiale du problème, l'AUV dans la zone de départ doit réaliser l'image SAS des trois sous-zones de mission. 45
3.8 Représentation de la structure des objets de la mission exemple. 46
3.9 Représentation des intervalles sur lesquels les trois types de conditions temporelles doivent être vérifiés. 52
3.10 Représentation des différentes contraintes possibles entre intervalles. . 53
3.11 Un exemple de problème dans la première catégorie de Cushing, toute solution est séquentielle. 57
3.12 Un exemple de problème dans la deuxième catégorie de Cushing, il admet des solutions séquentielles ou concurrentes. 58
3.13 Un exemple de problème dans la troisième catégorie de Cushing, toute solution est concurrente. 58
3.14 Exemple de deux actions nécessairement concurrentes. 59
3.15 Exemple de plan partiellement ordonné ainsi que quelques plans fixes qui lui sont associés. 60
3.16 Classification des approches de planification en fonction du type de problème résolu et du type de solution produit. 61
3.17 Principe de résolution dans l'état courant de tâches primitives et abstraites. 64
3.18 Les trois types de raffinement possibles dans un plan partiel. 67
3.19 Compilation de deux actions temporelles nécessairement concurrentes en une macro action représentant l'exécution des deux actions. 69
3.20 Classification des contributions réalisées. 74

4.1 Un plan en couches avec les états successifs résultant de l'application des couches. 79
4.2 État initial du problème encodé, la tâche initiale est introduite dans le premier taskholder, la couche courante est vide et il n'y a aucune contrainte d'ordre. 85

4.3 La tâche $T0$ est décomposée en quatre tâches, $T1$ et $T2$ sont abstraites, $t1$ et $no-op$ sont primitives. La tâche $no-op$ représente la dernière tâche du réseau. 86
4.4 Le taskholder $th2$ n'a pas de prédécesseur et contient une tâche primitive. L'action correspondante $a(t1)$ est ajoutée à la couche courante et le taskholder est marqué comme résolu. 86
4.5 La couche courante est terminée en la vidant et en réinitialisant les taskholders résolus. 87
4.6 Le *makespan* du plan de solution retourné par sCTHD, ainsi que le temps de résolution, en fonction du nombre de taskholders sur une instance de Satellite. 94

5.1 Principe général de TEP, la résolution se déroule en deux temps : un problème relaxé est généré puis résolu, dans un second temps, les contraintes relaxées sont réinjectées et la solution générale est générée en affectant des dates de début aux actions. 99
5.2 Schéma d'une action temporelle a. Elle est définie sur un intervalle $[v_a^s, v_a^e]$ et est composée de deux actions instantanée $start(a)$ et $end(a)$ ainsi que d'un ensemble d'invariants $inv(a)$ devant être vérifiés sur $]v_a^s, v_a^e[$. 101
5.3 Schéma de la représentation d'une action temporelle par deux actions instantanées et un lien causal. 103
5.4 Schéma d'une action non représentable par des actions instantanées classique. 104
5.5 Schéma de la représentation d'une action temporelle dans TEP à l'aide de deux évènements temporels et d'un lien causal. 105
5.6 Schéma de la représentation d'une tâche abstraite dans TEP à l'aide de deux évènements temporels, l'un abstrait, l'autre temporel. 105
5.7 Exemple de représentation d'un réseau de tâches temporelles dans TEP. Deux tâches sont représentées, l'une est une tâche abstraite t et l'autre est primitive a. Le début de t est contraint avant la fin de a. 106
5.8 À l'état initial, trois tâches sont représentées dans le réseau, soit huit évènements au total. e_I et e_G représentent les états initiaux et finaux, a_s et a_e sont les évènements à a, b_s et b_e à b et t_s et t_e à la tâche abstraite t.110
5.9 La tâche abstraite est décomposée en insérant le réseau d'évènements temporels associé à la méthode utilisée dans la décomposition. 110
5.10 Un défaut de précondition ouverte est résolu en ajoutant un lien causal couplé à une relation d'ordre avec un évènement produisant la propriété requise. 110

5.11 Première manière de résoudre une menace causale, l'évènement menaçant le lien causal est ordonné strictement avant le lien causal. 111
5.12 Seconde manière de résoudre une menace causale, le lien causal est ordonné de telle sorte à se terminer avant que, ou en même temps que, se présente la menace. 111
5.13 Un exemple de TTDG, la tâche initiale $T0$ peut être décomposée par deux méthodes $M0$ et $M1$. Les nœuds représentant les méthodes sont eux-mêmes reliés aux tâches introduites par l'application de la méthode.113

6.1 Principe général de TTC, le problème hiérarchique temporel initial est encodé en un problème HDDL via le principe de relaxation des durées. Le problème encodé est résolu à l'aide d'une planificateur HDDL, les contraintes de durée sont réinjectées après une étape optionnelle de déordonnancement. 123
6.2 Compilation d'une tâche abstraite par TTC : la tâche abstraite t est compilée en deux tâches, l'une abstraite et l'autre primitive. 126
6.3 Compilation d'une tâche primitive par TTC : l'action durative a est compilée en deux actions instantanées, a_s et a_e. 126
6.4 Compilation complète d'une action durative par TTC : des préconditions ont été ajoutées afin de garantir la préservation des invariants. 127

B.1 Le réseau de tâches initial est composé d'une unique tâche $T0$, elle est donc la seule à pouvoir être sélectionnée. 156
B.2 Une tâche sans prédécesseur est sélectionnée pour être résolue. Dans le cas présent, $t4$ a été sélectionnée parmi $\{t4, t2\}$. 156
B.3 La tâche $t4$ est résolue en insérant a_4 dans le layer courant, puis seule $t2$ peut être sélectionnée. 156
B.4 Si a_2 ne peut pas être résolue dans le layer courant, CPFD introduit un nouveau layer en appliquant les effets du layer construit et en retirant du réseau les tâches résolues. 156
B.5 Si a_2 et a_4 sont mutuellement indépendantes, $t2$ peut être résolue en insérant a_2 au layer en courant. 156
B.6 Lorsque toute les tâches sans prédécesseurs sont résolues, le layer courant est appliqué à l'état courant, les tâches résolues sont retirées et un nouveau layer introduit. 156

Liste des tableaux

4.1 Temps de résolution des encodages 92
4.2 Makespan des solutions renvoyées 92

5.1 Résultats des expérimentations pour les différentes configurations. . . 118

6.1 Résultats des expérimentations pour les différentes configurations. . . 131

Liste des listings

3.1 Définition PDDL des types de la mission. 46
3.2 Déclaration des prédicats du problème. 46
3.3 Déclaration des actions du problème. 47
3.4 Fichier d'instance associé au problème exemple (voir Figure 3.7). . . . 48
3.5 Déclaration des prédicats du problème. 49
3.6 Déclaration des méthodes du problème. 49
3.7 Fichier d'instance associée au problème exemple (voir Figure 3.7). . . 50
3.8 Fichier d'instance associée au problème exemple (voir Figure 3.7). . . 50
3.9 Déclaration d'une méthode temporelle HDDL2.1. 53

A.1 Fichier HDDL définissant le domaine associé à la chasse aux mines. . . 153
A.2 Fichier HDDL définissant une instance de problème de chasse aux mines.154

Première partie

Introduction et état de l'art

1 Introduction

Il y a peu d'environnements plus inhospitaliers pour des opérations humaines que le milieu marin. En effet, malgré la place centrale de la mer dans les activités, les échanges, mais aussi les conflits humains, on estime que 95 pourcent des fonds marins restent inexplorés. Opérer dans le milieu marin nécessite d'importants moyens matériels et humains, et cela ne va pas sans risques pour les acteurs de ces opérations. En particulier en ce qui concerne les opérations sous-marines, l'utilisation d'appareils autonomes n'a cessé d'augmenter ces dernières années. Les appareils autonomes présentent de nombreux avantages : (1) ils permettent d'éloigner les opérateurs humains des environnements dangereux, (2) ils opèrent de manière automatisée et sont généralement plus efficaces que les flottes opérées manuellement. Par exemple, lors des opérations de recherche du vol MH370 de la Malaysia Airlines, l'utilisation d'avions, de bateaux et de sous-marins autonomes a permis une couverture rapide de l'immense zone de recherche [LM14]. Les opérations marines et sous-marines sont également prépondérantes dans le milieu militaire. En particulier, l'émergence de guerre asymétrique a aussi augmenté l'utilisation de mines sous-marines dans le but de contrer une flotte ennemie. Un enjeu majeur de la guerre moderne est donc de pouvoir détecter et neutraliser les mines présentes sur une zone. À ce propos, la marine américaine considère la modernisation de ses systèmes anti-mines comme une priorité [Arc17]. Dans ce type d'opération, les flottes d'appareils sous-marins inhabités sont essentielles pour préserver la sécurité des opérateurs humains.

Pour autant, opérer une flotte d'appareils autonomes est un grand défi technique. En effet, les objectifs d'une opération marine peuvent être multiples et de nature très différente. Lors d'une mission de déminage, une zone de mission est divisée en sous-zones et trois processus distincts doivent être réalisés dans chaque sous-zone de la mission. La première étape est de *détecter* les menaces potentielles de la zone. La deuxième étape est d'*identifier* lesquelles de ces menaces sont effectivement des mines. Enfin, la dernière étape consiste à *neutraliser* toutes les menaces identifiées. Chacune de ces étapes requiert l'utilisation d'appareils aux capacités spécifiques : si la phase de détection implique des drones sous-marins équipés de scanner, la phase de neutralisation implique des drones de neutralisation kamikaze. De plus, d'autres appareils de surface peuvent récupérer et larguer les appareils sous-marins pour accélérer les déplacements de ces derniers. Une opération de déminage sur de

multiples zones et avec toute une flotte d'appareils implique donc des contraintes de coordination et de synchronisation entre ces appareils. Un des enjeux majeurs est donc de pouvoir planifier la mission : en d'autres termes, déterminer à la fois les actions qui doivent être réalisées par les éléments de la flotte, mais aussi le moment auquel ces actions doivent être réalisées, tout en respectant les contraintes de ressources, de temps ou d'environnement spécifiques à la mission. En d'autres termes, il faut déterminer pour chaque appareil l'ensemble d'actions qu'il doit réaliser ainsi que la date à laquelle il doit le faire. La planification de mission est donc une tâche difficile qui peut prendre du temps ou même entraîner des erreurs si elle est réalisée par un opérateur humain.

Dans ces conditions, la planification automatique de mission est un point essentiel à l'opération effective d'une flotte d'appareils autonomes [SBE08]. Elle peut être utilisée soit comme un outil de génération de plans ou comme un outil d'aide à la conception de plans [DN10]. Dans les deux cas, la planification automatique est une aide précieuse en termes de temps, mais aussi quant à la correction des plans de mission produits. En effet, il est bien plus efficace d'utiliser l'informatique pour vérifier qu'un plan est exécutable ou qu'un ensemble de contraintes est vérifié. Cependant, la planification automatique présente aussi des enjeux en raison de la faible explicabilité de la plupart des systèmes de planification automatique. Autrement dit, il est difficile pour un opérateur humain de comprendre dans quel but les actions ont été insérées dans le plan. Ce point-là est particulièrement important si l'on considère que les missions planifiées automatiquement peuvent impliquer des équipements stratégiques, voire des vies humaines. Il est donc primordial d'avoir un contrôle sur les actions réalisées et pour quel objectif elles le sont.

Le sujet de cette thèse porte donc sur la planification automatique de mission, c'est-à-dire l'ensemble des procédés algorithmiques permettant de produire un plan répondant à un objectif donné. Nous nous concentrerons en particulier sur les procédés avec une certaine explicabilité, point que nous préciserons par la suite. Nous présenterons également les contributions qui ont été apportées au domaine de la planification automatique au cours de cette thèse. Si les systèmes présentés par la suite peuvent être appliqués à n'importe quel problème de planification, nous garderons la planification de mission de déminage comme un exemple concret de problème de planification tout au long de cette thèse.

Les travaux réalisés dans le cadre de cette thèse ont donné lieu aux publications suivantes :

— An efficient HTN to STRIPS encoding for concurrent plans [CPF23a]

— On guiding search in HTN temporal planning with non temporal heuristics [CPF23b]

La suite de cette thèse est organisée de la manière suivante : le chapitre deux présente les enjeux et les contraintes des missions de déminage, le chapitre trois présente l'état de l'art des approches existantes. Les chapitres quatre, cinq et six présentent les différentes contributions apportées lors des travaux de cette thèse. Et enfin, le chapitre sept apporte une conclusion et propose des ouvertures pour la suite de ces travaux.

2 Contexte général d'une mission de déminage

Le minage et le déminage ont pris une importance croissante dans les guerres modernes, en premier lieu lors des deux guerres mondiales, où elles furent intensément utilisées dans la Manche et la mer du Nord [Nie14].

C'est à cette occasion que les mines se sont révélées être une arme d'une efficacité redoutable. Elles ne nécessitent pas de commande de mise à feu et ne mettent pas en danger la vie d'un opérateur humain. De surcroît, elles commettent de sérieux dégâts aux bâtiments ennemis, pouvant mener à la perte des navires et de leur équipage.

Les mines ont depuis été utilisées de nombreuses fois dans le cadre des guerres asymétriques. Elles sont en effet particulièrement efficaces lorsqu'un seul des deux belligérants possède une flotte navale dans le sens où elles permettent de causer d'importants dégâts pour un coût relatif bien moindre.

La problématique du déminage se pose ainsi aux armées possédant une flotte importante. L'objectif est donc d'être capable de détecter la présence de mines sur une zone et de les retirer dans le cas où la zone en question ne peut pas être évitée.

Ce chapitre est organisé de la façon suivante : nous présenterons d'abord les différents types de mines ainsi que leur fonctionnement. Puis, nous décrirons les systèmes utilisés dans les missions de déminage. Enfin, nous présenterons un exemple type de mission de déminage.

2.1 Différents types de mines

Les premières utilisations de mines marines remontent à la Chine du quatorzième siècle. À l'époque et jusqu'au début du dix-neuvième siècle, les mines marines étaient des engins flottants contenant des explosifs, dont la détonation se faisait manuellement ou par un système de détonation retardée par une mèche. La Figure 2.1 représente un croquis d'une mine utilisée lors de la guerre de Sécession américaine,

dans la seconde moitié du dix-neuvième siècle. Pour comparaison, la Figure 2.2 représente un modèle allemand utilisé lors de la Seconde Guerre mondiale.

Bien entendu, les types de mines ainsi que leur mode de fonctionnement ont fortement évolué avec le temps et la technique. Ces différents types de mines sont généralement classifiés dans trois catégories :

— La première catégorie de mine est celle des mines *de contact*. Comme leur nom l'indique, ces mines détonent lorsqu'un choc est appliqué sur la mine. Ce type de mines se décline en de nombreuses variations, certaines mines étant maintenues au raz de l'eau par un câble métallique (on parle d'un orin). D'autres flottent et dérivent simplement à la surface de l'eau dans l'espoir de trouver une cible ennemie. On parle alors de mines dérivantes.
— La deuxième catégorie de mine est celle des mines *à influence*. Elle comprend les mines dont le déclenchement se fait par l'action indirecte du navire sur la mine. La majorité des mines modernes sont comprises dans cette catégorie. On peut notamment citer les mines équipées de magnétomètres, qui détectent la déformation du champ magnétique terrestre engendrée par le passage d'un navire métallique de gros tonnage. D'autres sont équipées d'hydrophones, capables de repérer la signature acoustique produite par le moteur et les pales d'un navire. Enfin, certaines mines sont aussi capables de détecter les changements de pression engendrés par le passage d'un navire.
— La dernière catégorie de mines est celle des *mines télécommandées*. Comme leur nom l'indique également, ces mines sont équipées d'un détonateur qui permet à un opérateur d'activer ces mines le moment voulu, et ce à distance. Il faut également noter que ces mines sont souvent couplées d'un mode les rendant équivalentes à des mines de contact ou à des mines à influence régulière. Parmi ces mines on retrouve notamment des mines installées dans des ports ou des endroits stratégiques par temps de paix.

Outre leur mode de déclenchement, on peut également catégoriser les mines en fonction de leur position par rapport au fond marin. On distingue alors deux types de mines :

— Les mines *de surface*. Cela comprend toutes les mines positionnées à proximité de la surface de l'eau afin d'être au plus proche des bâtiments ennemis. Ces mines peuvent être soit maintenues en place à une position donnée par un orin, soit laissées à la dérive dans le courant. Même si la majorité des mines de surface sont aussi des mines de contact, cette catégorie peut également comprendre des mines à influence ou télécommandées.

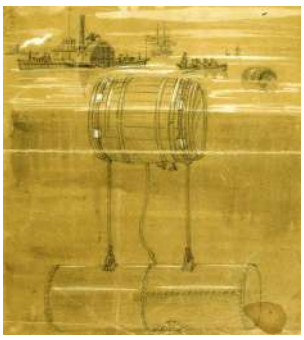

Figure 2.1. : Croquis d'une mine utilisée lors de la guerre de Sécession américaine en 1861.

Figure 2.2. : Mine allemande datant de la Seconde Guerre mondiale.

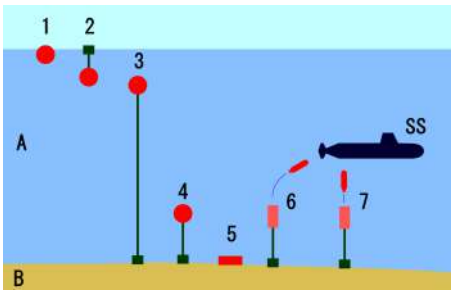

Figure 2.3. : Les différents types de mines en fonction de leur position dans l'eau. Les types 1 à 3 sont des mines de surface, quand les mines 4 à 7 sont des mines de fond.

— Les mines *de fond* sont des mines fixées au fond marin au moyen d'un leste ou à l'aide d'un orin (plus court que pour les mines de surface). Ces mines peuvent de même avoir tous les types de déclenchement, les mines de fond à influence visent principalement les navires de surface tandis que les mines de fond et de contact visent principalement les bâtiments sous-marins.

Un schéma des différents types de mines est représenté dans la Figure 2.3

2.2 Lutter contre les mines marines

Afin de lutter contre des mines en constante évolution, une branche entière du domaine militaire a été dédiée à la lutte contre les mines. On parle alors de *guerre des mines*. La guerre des mines a de nombreuses composantes. On peut par exemple citer la composante des renseignements, qui vise à déterminer quelles zones sont potentiellement minées. Parmi ces zones, celles qui ne sont pas essentielles à la poursuite de la mission sont classifiées comme zones à éviter. Cependant, pour toutes les zones essentielles potentiellement minées, il faut prévoir de neutraliser les

mines présentes. On parle alors de mission de *contre-mesure* (Mine Counter Measure Mission ou MCMM en anglais). C'est à ce type de mission que l'on s'intéressera par la suite.

Dans les missions modernes de contre-mesure, deux stratégies sont employées. La première stratégie est de simuler le passage d'un navire au-dessus de la zone minée afin d'activer les systèmes de déclenchement des mines. On réalise cette technique en tractant un leurre à l'arrière d'un autre véhicule. On parle alors d'opération de *dragage* des mines. La seconde stratégie vise à détecter individuellement les mines d'une zone afin de les neutraliser de manière ciblée. Cette stratégie requiert tout un ensemble d'agents, aux rôles distincts, pour mener à bien la mission. On parle alors d'opération de *chasse aux mines*. Ces deux stratégies ont chacune leurs avantages et inconvénients. Les deux parties suivantes détaillent chacune de ces stratégies.

2.2.1 Opérations de dragage des mines

La première des contre-mesures employées face aux mines est donc les opérations de dragage de mines. Historiquement, ces opérations furent les premières employées en raison de leur simplicité. Le principe général de ces opérations est de faire passer un leurre sur la zone minée afin d'activer à tort les systèmes de déclenchement des mines de la zone. Ce type d'opération ne peut donc uniquement neutraliser les mines dont le déclenchement peut être leurré, c'est-à-dire les mines à déclenchement automatique. On nomme ce leurre une *drague*. C'est un appareil habituellement tracté par un autre engin (un bateau ou un hélicoptère en général) moins sensible aux mines. On peut noter que les dernières utilisations militaires de navires à coque de bois furent pour des opérations de déminage. Ces navires n'ont en effet pas la signature magnétique ou sonore suffisante pour déclencher une mine destinée à un bateau en métal. Par la suite ont été préférés des engins de tractage volants (hélicoptères ou drones), pour plus de sécurité. Un exemple d'appareil tractant une drague est représenté sur la Figure 2.4. On compte deux catégories de drague, chaque catégorie correspondant à un type de déclenchement automatique de mine :

— *Drague mécanique :* ce type de drague vise à neutraliser les mines dont le déclenchement est mécanique, c'est-à-dire se déclenchant au contact d'un navire. Le principe de ces dragues est de sectionner l'orin maintenant la mine sous la surface de l'eau. La mine se met donc à flotter et peut être repérée et neutralisée par les navires de surface.
— *Drague à influence :* ces dragues cherchent à neutraliser les mines se déclenchant par l'action indirecte d'un bateau (les mines à influence). Les principaux

Figure 2.4. : Hélicoptère tractant une drague.

modes de déclenchement des mines à influence étant la reconnaissance magnétique ou sonore du passage d'un bateau, on compte aussi deux types de drague à influence : les dragues magnétiques et les dragues sonores. Chaque type de drague vise à neutraliser les mines avec un mode de déclenchement correspondant.

Les opérations de dragage présentent l'avantage majeur de ne pas avoir à se soucier de la détection *a priori* des mines. Le but ici est de forcer le déclenchement des mines. Cependant, ces opérations ont également des inconvénients : ces opérations manquent fortement de discrétion, ce qui peut être un enjeu majeur dans certaines situations et, par ailleurs, ces opérations peuvent se révéler dangereuses, notamment pour l'appareil tractant la drague. Enfin, ce type d'opération ne permet pas de fournir une estimation de la qualité du déminage : par définition, la drague leurre le système de déclenchement des mines, elle peut uniquement traiter les mines dont le système de déclenchement est connu et imitable. Or, les mines marines sont en constante évolution tant du point de vue du fonctionnement que du déclenchement. En ce sens, une opération de dragage ne permet pas de garantir la sécurité d'une zone, elle ne permet de neutraliser que des mines déjà connues.

2.2.2 Opérations de chasse aux mines

La deuxième stratégie dans les missions de contre-mesure est de réaliser une opération de *chasse aux mines*. L'objectif de ces opérations est de détecter puis de neutraliser individuellement les mines d'une zone. Ces opérations se divisent généralement en trois phases distinctes :

— *Phase de détection* : l'objectif de cette phase est de scanner le fond ou la surface de l'eau afin d'y repérer tous les objets pouvant potentiellement être des mines.
— *Phase d'identification* : l'objectif est de classifier les menaces potentielles repérées lors de la phase de détection en menaces réelles ou en fausses alarmes.
— *Phase de neutralisation* : une fois les menaces classifiées, la phase de neutralisation vise à mettre hors d'usage les mines repérées.

La stratégie de la chasse aux mines est généralement préférée aux opérations de dragage. En effet, les opérations de chasse aux mines permettent d'avoir un meilleur contrôle de la neutralisation des mines. Contrairement aux opérations de dragage, la chasse aux mines permet tout autant de neutraliser les mines au fonctionnement déjà connu, que les mines encore inconnues. Ces opérations permettent également de choisir la *manière* et le *moment* pour la neutralisation des mines, ce qui permet de répondre aux enjeux de discrétion que peuvent présenter les opérations militaires.

Cette stratégie de contre-mesure est communément adoptée par les armées modernes, c'est donc celle que nous étudierons et prendrons pour exemple par la suite.

2.3 Déroulement d'une mission de chasse aux mines

Dans cette partie, nous décrivons un exemple de mission de déminage, dans le cadre d'une opération de chasse aux mines. L'objectif de cette partie est de rendre compte des enjeux et des contraintes qui peuvent intervenir dans une opération de déminage.

Comme énoncé précédemment, une opération de chasse aux mines est décomposée en trois phases : une phase de *détection*, une phase d'*identification* et une phase de *neutralisation*. Cette partie est donc organisée comme suit, nous définirons dans un premier temps la zone dans laquelle se déroulera la mission, puis nous détaillons successivement les trois phases d'une mission de chasse aux mines, enfin nous présentons un exemple de flotte moderne de chasse aux mines.

2.3.1 Définition de la zone de mission

Une des entrées d'une mission de déminage est la zone de mission. Cette zone se définit comme un ensemble de polygones. Chaque sous-zone correspondra à ce que l'on

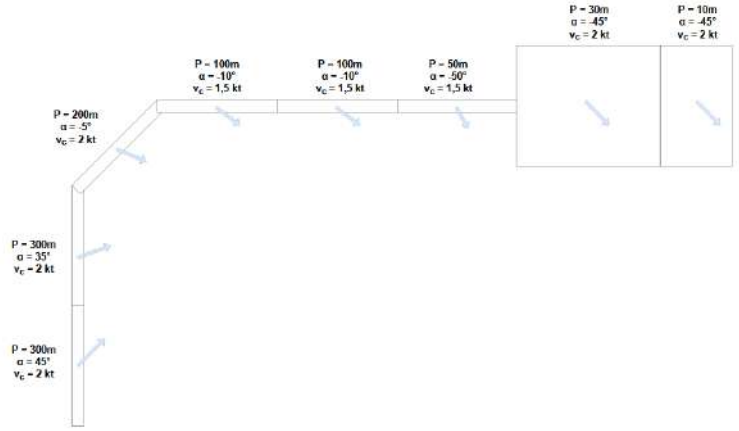

Figure 2.5. : Un exemple de zone de mission, la zone est divisée en sous-zones, les caractéristiques environnementales sont supposées constantes sur chaque sous-zone.

appellera une sous-zone de mission. Nous supposons que les caractéristiques environnementales (profondeur, température, courant, etc...) sont constantes sur chaque sous-zone. Une mission de déminage pouvant se dérouler sur plusieurs heures, ceci est une simplification du problème réel, mais une simplification nécessaire pour avoir un problème de taille raisonnable.

Définition 1 *Une **zone de mission** $S = \{s_1, s_2, \ldots, s_n\}$ est un ensemble de sous-zones $s_i = (\chi_i, e_i)$ avec $i \in [\![1, n]\!]$, où χ_i représente les coordonnées géographiques de la sous-zone et e_i ses propriétés environnementales.*

Un exemple de zone de mission est représenté sur la Figure 2.5. Dans cette mission, il s'agit de sécuriser un couloir d'approche ainsi que deux zones d'opérations. Il y a donc six sous-zones représentant le couloir d'approche ainsi que deux sous-zones, plus larges, représentant les zones d'opérations. Chaque sous-zone possède trois caractéristiques environnementales : la profondeur du fond P (en mètres), l'orientation du courant α (en degrés) ainsi que la vitesse du courant v_c (en nœuds).

Il faut noter que définir ces sous-zones est une question en soit. En effet, il faut définir des zones suffisamment grandes pour accueillir la flotte, tout en cherchant à minimiser l'espace à déminer, et ce, en essayant d'avoir des sous-zones sur lesquelles les propriétés environnementales sont homogènes. L'acquisition de ces données environnementales se fait lors d'une phase appelée REA (Rapid Environmental Assessment). Un exemple d'une telle mission est illustré à la référence suivante [Sim+11]. Cependant, ce problème ne sera pas traité ici et par la suite on considérera

Figure 2.6. : Un A18 de la société Exail Robotics

que les sous-zones de mission ainsi que leurs propriétés sont des données d'entrée du problème à résoudre.

2.3.2 Phase de détection

La première phase d'une mission de chasse aux mines est donc de détecter les objets de la zone pouvant être des mines. Pour cela, on utilise des drones sous-marins équipés de sonar afin de réaliser une image du fond marin. La méthode moderne pour faire ce type d'image repose sur la technologie du Sonar à Antenne Synthétique (SAS). Cette méthode repose sur le principe de créer une antenne virtuelle en réalisant des émissions successives le long d'un rail. Appliqué au milieu sous-marin, cela correspond à faire naviguer un drone équipé d'émetteurs et de récepteurs basses fréquences le long d'une trajectoire linéaire sous-marine [AHL07]. La Figure 2.6 représente une photo d'un appareil A18 de la société Exail Robotics, capable de réaliser de telles images SAS.

Le suivi de cette ligne sous-marine permet de créer une image du fond présent sous l'appareil. Afin d'obtenir une image globale de la zone, il faut donc la parcourir en suivant un ensemble de rails bien placés et d'assembler bout à bout les images obtenues de chaque rail. Un exemple simple de rails sous-marins recouvrant une zone est représenté sur la Figure 2.7.

Le placement de ces lignes de trajectoire va jouer un rôle majeur dans la qualité des images produites. En effet, la résolution des images dépend majoritairement de (1) la longueur de l'antenne synthétique créée et de (2) la précision avec laquelle la trajectoire est suivie. Or la précision de suivi d'une trajectoire dépend quant à elle de l'orientation de cette trajectoire par rapport au courant sous-marin. Plus le courant frappe l'appareil de manière latérale et plus il le fait dériver dans cette même direction latérale. Cette dérive latérale perturbe le suivi d'une trajectoire en ligne droite. On préférera donc les lignes de trajectoire les plus colinéaires au courant possible. Bien sûr, naviguer colinéairement au courant implique dans un sens, de

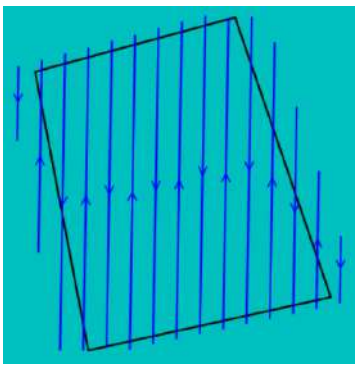

Figure 2.7. : Rails à suivre par l'appareil sous-marin pour cartographier la zone.

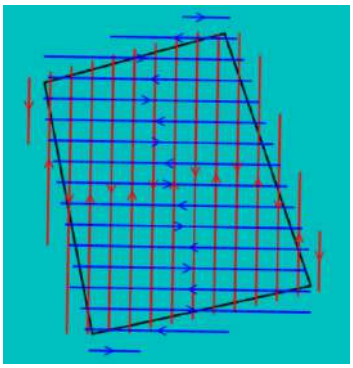

Figure 2.8. : Un exemple de rails double, permettant deux prises de vue de chaque point.

bénéficier de la poussée du courant, mais dans l'autre, de devoir lutter pour pouvoir le remonter. Plus la navigation est colinéaire au courant, et plus cet effet est fort. L'objectif est donc de chercher à optimiser la qualité des images produites tout en respectant les contraintes liées aux appareils à disposition ou à l'environnement.

Une autre stratégie pour augmenter la qualité des images finales est de multiplier les prises de vue. Cela signifie réaliser plusieurs images d'une même zone, en utilisant des ensembles de rails différents pour chaque image. Les images sont ensuite fusionnées pour obtenir une image finale de meilleure qualité. Un exemple de ces ensembles de rails est illustré à la Figure 2.8.

En fonction de la qualité d'image recherchée, il faut donc trouver le bon compromis entre le placement des rails, le nombre de prises de vue et le temps requis pour l'opération globale.

Finalement, on obtient des images du fond marin comme représenté dans la Figure 2.9. Une fois ces images réalisées, elles sont traitées afin d'y repérer d'éventuels objets suspects. C'est l'objet de la phase d'identification qui sera détaillée dans la partie suivante.

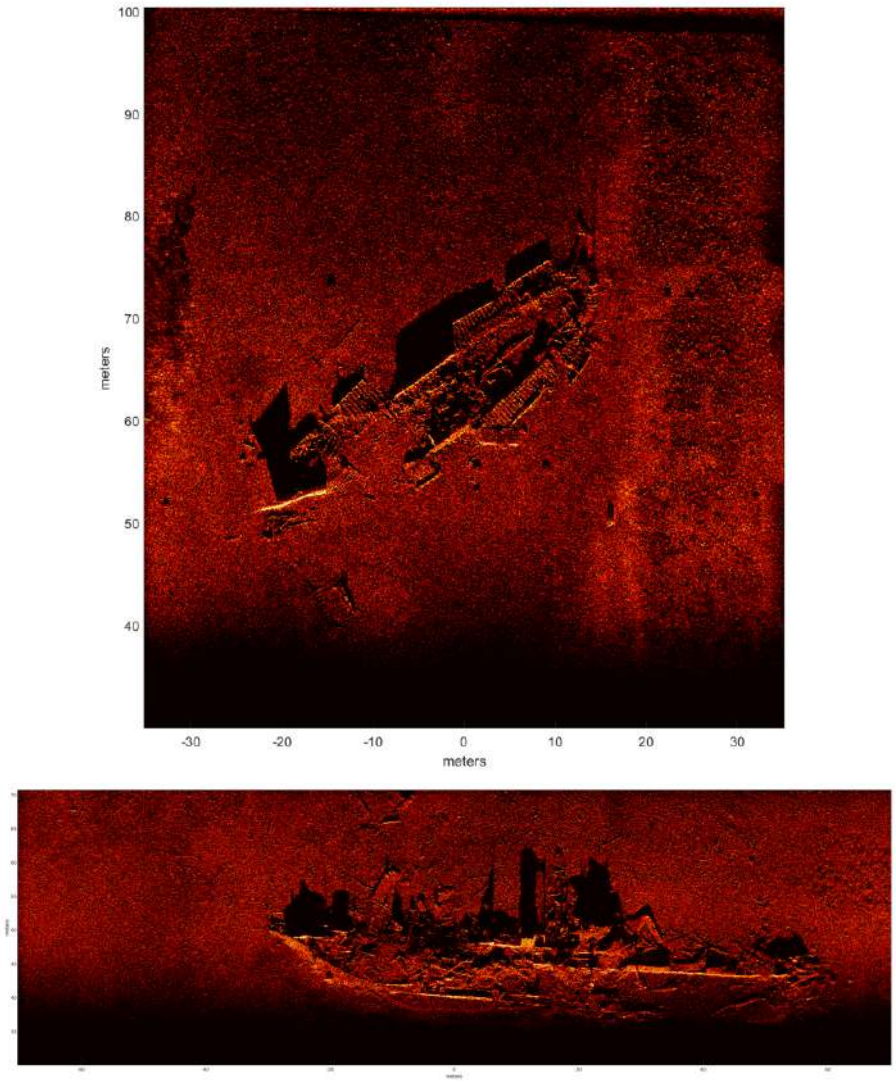

Figure 2.9. : Deux images SAS d'épaves dans la baie de Hyères.

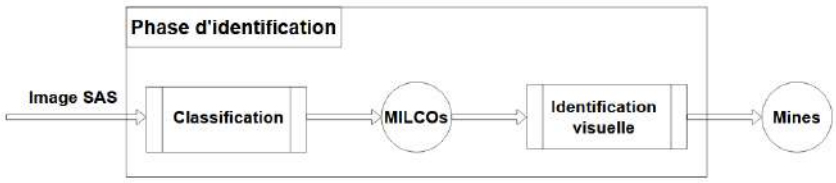

Figure 2.10. : Les étapes d'une phase d'identification classique.

2.3.3 Phase d'identification

La phase d'identification prend comme entrée l'image de la zone de mission réalisée lors de la phase de détection. L'objectif de cette phase est de déterminer si des mines sont présentes sur la zone et, si oui, de déterminer leur position.

Cette phase de la mission est habituellement divisée en deux étapes. La première étape consiste à analyser l'image de la zone de mission. De cette analyse, sont déduits un ensemble d'objets ressemblant à des mines, on parle de MILCO (MIne Like COntacts). On nomme cette étape la *classification*. La deuxième étape consiste à établir un contact visuel entre ces MILCO et un opérateur humain afin de déterminer s'il s'agit ou non d'une véritable mine. On appelle cette étape, l'*identification visuelle*. La Figure 2.10 schématise les étapes de la phase d'identification.

Les méthodes pour extraire les MILCO de l'image SAS sont nombreuses et ont suivi les avancées faites dans le domaine de la reconnaissance d'images. Historiquement, les méthodes employées cherchent à reconnaître des formes particulières de mine au moyen de masques de convolution. On peut soit essayer de reconnaître la forme de la mine directement [Flo+03], ou de reconnaître la forme de son ombre [Qui+05]. Plus récemment, les réseaux neuronaux ont également pu être utilisés pour reconnaître la forme d'une mine sur une image SAS [Ge+21]. Il faut rappeler que l'objectif premier de la classification est de classifier correctement toutes les mines de la zone comme MILCO, et non pas que tous les objets classifiés comme MILCO soient effectivement des mines. En d'autres termes, le point le plus important est d'être exhaustif sur les mines de la zone, quitte à classifier comme MILCO des objets qui ne sont pas des mines. L'objet de la deuxième étape d'une phase d'identification est précisément de chercher la classification visuelle par un opérateur humain de ces MILCO, en mines ou non-mines.

Cette seconde étape d'identification visuelle peut donc se faire par deux moyens, le premier est d'envoyer un plongeur directement auprès de la mine pour la classifier. Cette méthode est évidemment dangereuse pour le plongeur, et de plus certaines mines peuvent être difficiles d'accès (en eaux profondes, notamment). La doctrine

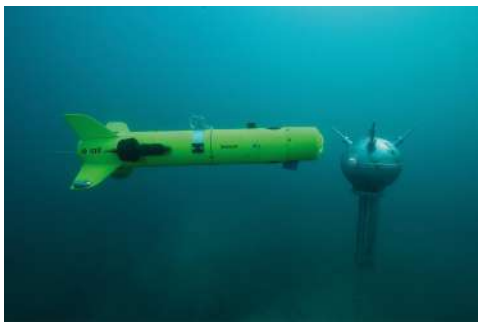

Figure 2.11. : Un appareil SeaScan d'Exail Robotics fournissant un contact visuel avec le MILCO.

moderne consiste donc à utiliser un autre type de drones sous-marins, équipés de caméras optiques, se rendant au plus près du MILCO afin d'en fournir l'image, à distance, à un opérateur humain. La Figure 2.11 représente un de ces drones sous-marins, le SeaScan de Exail Robotics.

À la suite de la phase d'identification, un ensemble de mines a été identifié sur la zone de mission. La phase suivante consiste à neutraliser les mines identifiées. Ce sera l'objet de la partie suivante.

2.3.4 Phase de neutralisation

La phase de neutralisation vise à retirer aux mines de la zone leur capacité de nuisance. Pour ce faire, deux méthodes sont possibles. La première est de désamorcer la mine. C'est une opération compliquée et dangereuse pour l'opérateur la réalisant. Cette méthode est beaucoup utilisée dans les opérations terrestres dans le but de préserver l'environnement autour de la mine. Dans le cadre du déminage marin, la phase de neutralisation consiste à forcer l'explosion de la charge de la mine.

Pour cela, il existe deux stratégies principales. La première est de déposer une charge explosive à proximité de la mine pour ensuite la faire détoner à distance. Cette tâche peut encore être réalisée par des plongeurs humains, ce qui présente d'évidents problèmes de sécurité. Avec l'apparition des systèmes autonomes, la tendance est d'utiliser ces appareils afin d'éloigner l'humain du danger. La Figure 2.12 représente un PAP (Poisson Auto Propulsé) qui est le drone sous-marin historiquement utilisé par la marine française pour la neutralisation de mines. Cet appareil dépose la charge maintenue sur sa partie inférieure, il est ensuite récupéré sur le navire. Dans un dernier temps, la charge est mise à feu pour neutraliser la mine.

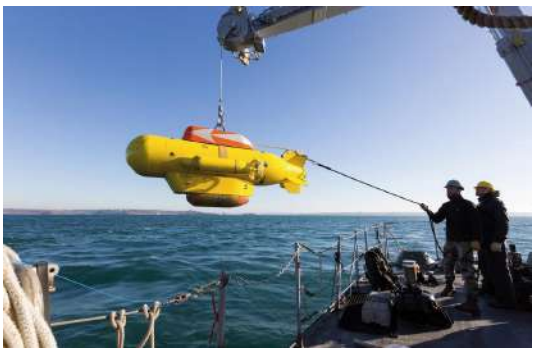

Figure 2.12. : Un Poisson Auto Propulsé (PAP), appareil autonome de neutralisation de mines.

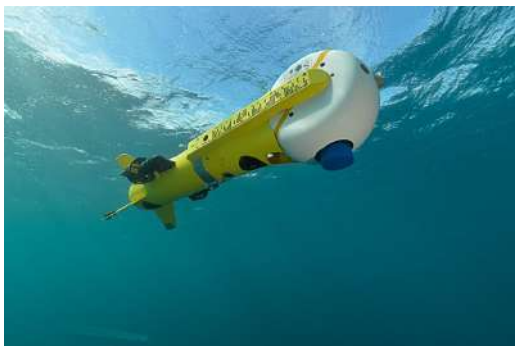

Figure 2.13. : Un drone KSter de Exail Robotics.

La seconde stratégie est d'utiliser le drone lui-même pour faire détoner la mine. On parle alors de drone kamikaze. Un exemple de ce type d'appareil est le drone KSter d'Exail Robotics, représenté dans la Figure 2.13. Ce drone est équipé d'une charge explosive à l'avant de l'appareil. Son objectif est de venir au contact direct de la mine pour faire détoner sa propre charge et la mine conséquemment.

2.3.5 Flottes intégrées MCM (Mines Counter Measures)

Nous avons donc vu les différentes phases d'une mission de déminage ainsi que les solutions pour accomplir chacune d'entre elles sur une sous-zone de mission. Dans la pratique, les opérations de chasse aux mines utilisent toute une flotte d'appareils opérant simultanément. L'intérêt est de pouvoir paralléliser la réalisation des différentes phases de mission sur les sous-zones : la neutralisation des mines d'une sous-zone peut être réalisée durant la détection d'une autre sous-zone. De plus, ces flottes d'appareils intègrent aussi des appareils de soutien qui assurent le bon déroulement de la mission ou permettent d'en augmenter son efficacité. L'ensemble de ces appareils forme une flotte intégrée MCM.

Figure 2.14. : Vaisseau-mère M940 (Naval Group)

Figure 2.15. : Appareil de surface autonome de support Inspector 125.

Un exemple d'une telle flotte a été présenté par le programme conjoint de Naval Group et d'Exail Robotics lors de la modernisation des marines belges et néerlandaises. Dans ce programme, six types d'appareils différents sont considérés :

— *Vaisseau-mère* : cet appareil de surface contient les autres appareils au début de la mission. Il sert également à la classification des images SAS et à la définition des MILCO lors de la phase d'identification. Une représentation du M940 produit par Naval Group est affichée dans la Figure 2.14.

— *Drones de soutien de surface* : ces appareils assurent le soutien des autres appareils de la mission en larguant et récupérant les drones réalisant les phases de détection et/ou d'identification. Ils peuvent également servir de relais de communication entre les appareils. Dans ce programme a été proposé l'Inspector 125 d'Exail Robotics (Figure 2.15).

— *Drones de détection* : comme leur nom l'indique, ces drones ont pour objectif de réaliser la phase de détection de la mission. Pour cette tâche ont été proposés les drones A18 d'Exail Robotics (Figure 2.6).

— *Drones d'identification* : le programme a proposé les drones SeaScan d'Exail Robotics pour la réalisation de cette tâche (Figure 2.11).

— *Drones de neutralisation* : pour la neutralisation des mines identifiées, les drones KSter d'Exail Robotics ont été proposés (Figure 2.13).

Réussir à coordonner ces appareils est à la fois un enjeu et un défi. C'est un enjeu dans le sens où obtenir une coordination efficace entre les appareils permet d'obtenir de meilleurs plans de mission au niveau de la durée ou de l'utilisation de ressources. Cela représente également un défi, car coordonner l'action de toute une flotte d'appareils agissant simultanément génère des contraintes et donc des problèmes complexes à résoudre.

La coordination *à l'avance* de cette flotte est dénommée la *planification*. L'objet de cette thèse porte sur ce sujet, l'objectif est de développer un système capable de produire un plan de mission pour une flotte intégrée de chasse aux mines.

2.4 Définition du problème et Ambitions

Dans ce chapitre, nous exposons le problème qui sera résolu au cours de cette thèse. Pour ce faire, nous commençons par exposer les suppositions qui sont faites ainsi que le problème à résoudre. Dans un deuxième temps, nous expliciterons nos objectifs quant à ce problème.

2.4.1 Hypothèses et problème

Hypothèse sur les données d'entrées du problème

Une mission de déminage se définit au moyen de deux éléments : (1) une zone de mission comme défini dans la Section 2.3.1 et (2) une flotte d'appareils disponible pour la mission. Si la zone de mission a été précédemment définie comme un ensemble de sous-zones couplées à un ensemble de propriétés environnementales, il convient de définir la flotte de la mission.

Une flotte de chasse aux mines est caractérisée par les appareils la composant. Ces appareils sont eux-mêmes caractérisés par un ensemble de propriétés. Ces propriétés comprennent des informations comme le type d'appareil, la position, l'état de ses différents modules, ou la charge de la batterie. Il faut noter que certaines de ces propriétés sont fixes (comme le type de l'appareil) et certaines peuvent être modifiées au cours de la mission (comme la position de l'appareil ou sa batterie). On a donc la définition suivante :

Définition 2 *Une **flotte** $F = \{ap_1, ap_2, \ldots, ap_n\}$ est un ensemble d'appareils $ap_i = (p_{i,1}, p_{i,2}, \ldots, p_{i,k})$ avec $i \in [\![1, n]\!]$, où $p_{i,j}$ représente la propriété j de l'appareil i.*

Hypothèses sur les agents et la zone de la mission

Comme cela a été exprimé lors du chapitre précédent, une mission de chasse aux mines peut comprendre de nombreux appareils différents. Dans la suite, nous considérons des flottes intégrées semblables à celle présentée par le programme conjoint Naval Group/Exail Robotics. Nous considérons donc les flottes qui comprennent les appareils présentés précédemment : *vaisseau-mère*, *drones de soutien de surface*, *drones de détections*, *drones d'identification* et *drones de neutralisation*. Nous faisons les suppositions suivantes sur ces appareils :

— *Vaisseau mère :* cet appareil sera supposé unique, immobile et le point de départ des autres appareils de la mission.
— *Drones de soutien de surface :* on supposera que ces appareils peuvent embarquer, mettre à l'eau et récupérer les drones de la mission, soit les drones de *détection*, d'*identification* et de *neutralisation*.

On considère les flottes comprenant une *dizaine d'appareils* au maximum. Cela correspond à la taille d'une flotte de chasse aux mines standard dans une armée moderne.

On suppose que la zone de mission, ainsi que sa décomposition en sous-zones est une entrée du problème. On suppose également que l'on possède les propriétés environnementales associées à chaque sous-zone, comme défini dans le paragraphe 2.3.1. On considère des missions dont le nombre de zones est d'une dizaine au maximum également, afin de garder des scénarios crédibles et des problèmes de complexité raisonnable.

Hypothèses sur les modules disponibles

Une mission de déminage maritime comprend de nombreux appareils hétérogènes avec chacun leurs contraintes et objectifs. Modéliser et résoudre directement un problème de planification de mission sous sa globalité, tel énoncé, serait d'une extrême complexité. Afin de répondre à cette limitation, des modules spécialisés pour la résolution de sous-problèmes sont utilisés. Par exemple, la planification des trajectoires des appareils est réalisée au moyen d'un module de planification de trajectoire dédié. Dans ce cadre, la planification consiste à déterminer *ce qui doit être fait*, tout en laissant la réalisation concrète de l'action planifiée au module dédié. Par la suite on suppose que les modules suivants sont à notre disposition :

— *Module de planification de trajectoire :* Ce module définit la trajectoire que doit suivre un appareil pour se rendre d'un point A à un autre point B. On supposera que ce module peut définir les trajectoires de n'importe quel appareil de la flotte (appareils aériens, sous-marins ou de surface).

— *Module de placement de rails :* Comme cela a été exposé dans le paragraphe 2.3.2, la phase d'identification requiert le placement puis le suivi de rails pour la création d'images SAS de la zone de mission. On supposera que ce module est capable de (1) placer les rails de mission et de (2) calculer le temps total que prendra un appareil pour suivre ces rails.

— *Module de détection de MILCO :* Ce module permet de détecter les MILCO présents sur une image SAS formée lors d'une phase d'identification. On supposera que ce module renvoie la liste des coordonnées géographiques des MILCO détectés.

Hypothèse du monde fermé

Notre approche est une approche *hors ligne*, ce qui signifie que l'on cherche un plan de mission à partir d'une situation initiale connue. Dans ce cadre, une connaissance *totale* et *exacte* de l'environnement est supposée. Nous supposons également que les actions réalisées par les agents de la mission sont *déterministes*, en d'autres termes que l'on peut prévoir le résultat d'une action de manière certaine.

Enfin, nous faisons la supposition que nous avons une connaissance totale de l'état de la zone de mission et de la flotte à disposition de la mission.

2.4.2 Ambitions

Le sujet de cette thèse est de proposer une approche permettant de planifier une mission de chasse aux mines dans le cadre défini précédemment. Planifier une mission revient à répondre aux deux questions suivantes : (1) quelles *actions* doivent être réalisées pour accomplir la mission et (2) à quels *moments* ces actions doivent être réalisées. Planifier une mission de chasse aux mines signifie donc produire pour chaque appareil une séquence d'actions à réaliser, ainsi que la date à laquelle chaque appareil doit le faire. Notre objectif est de proposer un système capable de produire des plans de mission de manière automatique, on parle alors de *planification automatique*. Le planificateur proposé doit également répondre aux enjeux suivants :

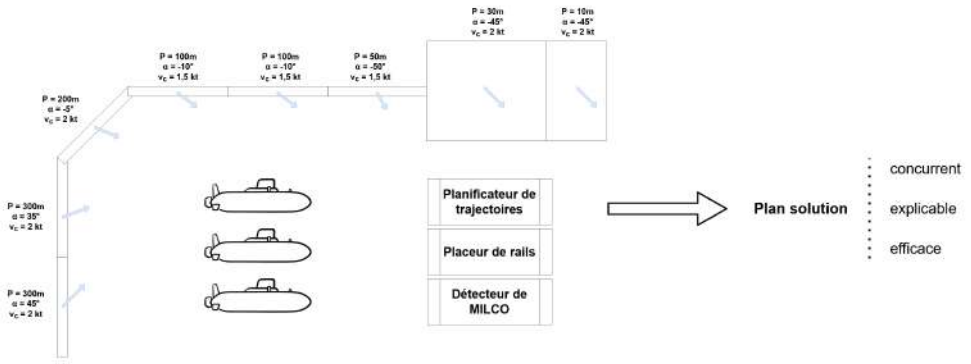

Figure 2.16. : Vue récapitulative du système recherché dans le cadre de cette thèse.

— *Efficacité* : l'objectif est de générer les plans les plus optimisés en termes de durée. Autrement dit, on cherche à réaliser la mission dans le délai le plus court. En particulier, une des clefs de l'efficacité est de produire des plans avec un haut degré de concurrence : c'est-à-dire des plans ou les actions des appareils se déroulent simultanément. L'approche proposée porte une attention particulière sur ce point.

— *Explicabilité* : le résultat du système proposé, que l'on nommera un *plan solution* par la suite, doit être lu et interprété par un opérateur humain. Le système doit donc être capable d'expliquer dans quel but chaque action est prise.

— *Expressivité et Modularité* : le monde de la guerre des mines étant en constante évolution, le système proposé se veut suffisamment expressif et modulable pour pouvoir être utilisé dans d'autres contextes que dans le cadre défini au chapitre précédent.

Un schéma récapitulatif du système recherché est représenté dans la Figure 2.16.

De nombreux travaux ont été réalisés dans le domaine de la planification automatique. Le sujet de la prochaine partie est de donner un compte-rendu des approches existantes ainsi que de leurs spécificités.

3 État de l'art

Dans ce chapitre, nous proposons de dépeindre un aperçu des différentes techniques de planification employées dans le domaine de la chasse aux mines et dans celui de la planification de missions sous-marines en général. La suite du chapitre est organisée de la manière suivante : dans un premier temps, une classification des systèmes de planification de missions de chasses aux mines est proposée. Ces systèmes sont classifiés en fonction du type de résolution employé. Quatre types de résolution sont identifiés et sont associés avec les systèmes de planification correspondants. La seconde partie de ce chapitre met l'accent sur un de ces types de résolution, celui qui a été choisi dans la suite de cette thèse. L'objectif est de détailler cette méthode de résolution particulière et d'en présenter les techniques.

3.1 Planification de mission pour une flotte maritime intégrée

Dans la littérature, un grand nombre de systèmes ont été proposés afin de planifier une mission de chasse aux mines. Ces systèmes sont de natures différentes et répondent à des problématiques qui ne sont pas toujours identiques d'un système à l'autre. On peut cependant classifier ces systèmes en deux grandes catégories.

La première catégorie est composée des systèmes reformulant la mission de chasse aux mines en un problème dont on connaît des techniques de résolution efficaces. Nous appellerons cette catégorie les systèmes par *reformulation*. Parmi les systèmes existants, la mission de chasse aux mines est reformulée soit en un *problème de tournées de véhicules* (ou *VRP* pour *Vehicule Routing Problem*) [TV14], soit en *problème de satisfaction de contraintes* (ou *CSP* pour *Constraint Satisfaction Problem*) [Mig03]. Si cette catégorie de systèmes bénéficie de l'efficacité des techniques de résolution des problèmes VRP ou CSP, elle souffre aussi de devoir exprimer la mission de chasse aux mines sous cette forme. Cela nécessite souvent de devoir modifier ou étendre les techniques de résolution existantes pour les adapter à notre cas particulier, comme nous le verrons par la suite. Toute modification des caractéristiques des appareils utilisés dans la flotte de mission implique une modification de la modélisation du

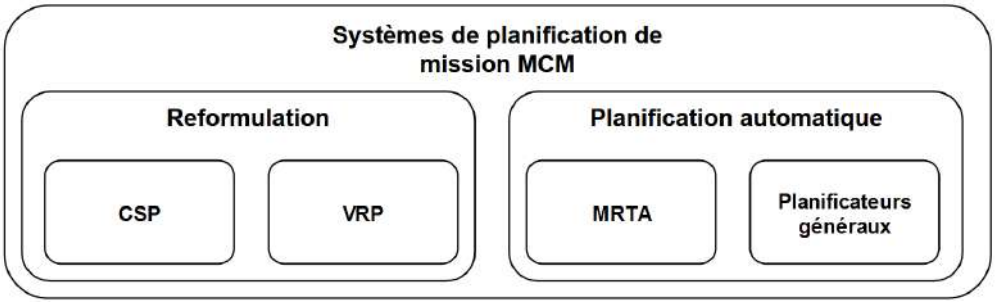

Figure 3.1. : Représentation des différentes catégories de systèmes dans le contexte de la chasse aux mines.

problème de chasse aux mines en VRP ou CSP. Ainsi, les approches par reformulation sont généralement efficaces, mais sensibles à tout changement des paramètres du problème. Il faut également noter que d'autres problèmes classiques peuvent être utilisés pour des reformulations. Par exemple, [Sch21b] propose une reformulation en problème de satisfaisabilité booléenne (*SAT* pour Boolean satisfiability problem [Coo23]). Cependant, la proximité du problème de chasse aux mines avec les problèmes CSP et VRP fait que ceux-ci sont préférés pour la reformulation.

Les systèmes de seconde catégorie ont une approche plus générale de la planification de mission. Ces approches cherchent à résoudre un problème général de planification, qui ne dépend pas d'un contexte particulier. Nous appellerons cette catégorie les approches *planification automatique* (*Planning and Scheduling* en anglais) [GNT04]. Ces approches sont suffisamment générales pour être appliquées à plusieurs contextes d'application. Parmi ces approches, deux stratégies sont employées. La première est de diviser la mission en un ensemble de sous-objectifs, qui sont ensuite distribués entre les agents de la mission. On parle alors de problème d'allocation de tâche multi-robots (ou *MRTA* pour *Multi-Robot Task Allocation*). La seconde stratégie est de planifier la mission dans son ensemble, pour trouver une solution à la mission globale sans avoir à la diviser en sous-objectifs. On parlera alors de *planificateur généraux*.

Un schéma représentant les différentes catégories de systèmes de planification est représenté dans la Figure 3.1.

À ce propos, certains systèmes de planification de missions de chasse aux mines incluent un système de *replanification*. Soit précisément le système cherchant à réparer un plan qui ne peut plus s'exécuter pour quelques raisons. Ce point sera précisé au cas par cas.

La suite de ce paragraphe est organisée comme suit, une première partie présente les approches par *reformulation* et la seconde présente les approches de planification automatique.

3.1.1 Approches par reformulation

Comme cela a été exposé précédemment, les approches par reformulation se divisent en deux catégories de méthodes de résolution. La première catégorie reformule le problème de chasse aux mines en un problème de satisfaction de contraintes, la seconde reformule le problème de chasse aux mines en un problème de VRP. Les deux parties suivantes sont respectivement consacrées à ces deux catégories de méthode.

Planification par reformulation CSP

Une première façon de planifier une MCM est donc de représenter la mission comme un problème d'optimisation combinatoire où la solution respecte l'ensemble de contraintes à résoudre. Dans ces approches, chaque appareil de la flotte est représenté par un ensemble de variables pouvant prendre des valeurs dans un domaine prédéfini. Ces variables sont également liées entre elles par un ensemble de contraintes modélisant à la fois l'environnement et les interactions des appareils avec ces derniers et entre eux. Le but du solveur est donc d'affecter des valeurs à ces variables de sorte à satisfaire les contraintes à tout instant de la mission.

Plus formellement on définit un problème de satisfaction de contraintes (ou CSP) de la manière suivante [RN10] :

Définition 3 *Un **problème de satisfaction de contraintes** $P = (X, D, C)$ est composé d'un ensemble de variables $X = \{X_1, X_2, \ldots, X_n\}$, un ensemble de domaines non vides $D = \{D_1, D_2, \ldots, D_n\}$, et d'un ensemble de contraintes $C = \{C_1, C_2, \ldots, C_m\}$.*

Chaque contrainte $C_i \in C$ est une paire $\langle t_i, R_i \rangle$, où $t_i \subseteq X$ est un sous-ensemble de variables de X et R_i une relation portant sur les variables de t_i. Cette relation peut être définie soit comme l'ensemble des valeurs possibles pour les variables de t_i, soit comme une formule liant les variables de t_i.

Une solution de P est une affectation des variables de X à des valeurs de leurs domaines respectifs de D tout en satisfaisant les contraintes de C.

Ces approches possèdent une très grande généralité permettant de modéliser un grand nombre de phénomènes différents. Par exemple, Yilmaz et al. [Yil+08] propose une approche permettant de planifier les trajectoires d'une flotte d'appareils sous-marins en fonction de la profondeur du fond, du vent, de la proximité des autres appareils ou encore de la portée des communications. Tous ces éléments sont modélisés par un ensemble de contraintes linéaires dont les variables modélisent l'environnement et les appareils de la mission. Cette approche présente l'avantage de n'utiliser que des contraintes linéaires et donc d'utiliser les puissants solveurs spécifiques à ces contraintes. Dans ce cas, un algorithme de type séparation et évaluation (Branch and Bound) est utilisé pour la résolution.

Un autre exemple d'approche par satisfaction de contraintes a été proposé dans [Gen+13]. La particularité de cette approche est de diviser la recherche en deux temps, un premier où une solution initiale est recherchée, et un second où la solution initiale est optimisée en fixant le nombre d'appareils utilisés. Contrairement à l'approche précédente, celle-ci utilise un algorithme génétique [Mit98] couplé à un algorithme de colonies de fourmis [DG97] pour trouver une solution au problème.

Un dernier exemple d'utilisation de problème de satisfaction de contraintes pour la planification de flottes d'AUV (pour *Autonomous Underwater Vehicules*) a été proposé par McGann et al. [McG+08]. Dans cette approche, le problème de satisfaction de contraintes permet de déterminer les actions à entreprendre pendant le déroulement même de la mission. Cette approche permet donc de s'adapter en temps réel aux contraintes environnementales. Cependant, cette approche se limite au contrôle d'un unique AUV et ne permet pas de gérer les interactions des appareils entre eux.

Approches par reformulation VRP

Le problème de tournées de véhicules (VRP pour Vehicule Routing Problem) [DR59] est un problème du domaine de la recherche opérationnelle dont le but est de déterminer les trajectoires d'un ensemble de véhicules de sorte à passer par un ensemble de points définis a priori. Il s'adapte particulièrement aux missions de chasse aux mines, où la zone de mission est divisée en sous-zones à visiter par les appareils afin d'y réaliser les différentes tâches de la mission. Ainsi, la reformulation VRP est un choix populaire pour représenter une mission de chasse aux mines ou pour le contrôle d'appareils sous-marins.

Plus formellement, on peut définir un problème VRP de la façon suivante :

Définition 4 *Un **problème de tournées de véhicules** $P = (G, T)$ est composé d'un graphe orienté G et d'un ensemble de véhicules T. Le graphe $G = (V, E)$ est classiquement composé d'un ensemble de sommets V et d'arêtes E. Chaque arête $e = (a, b) \in E$ relie deux sommets a et b de V et est associée à un coût $c \geq 0$ représentant le coût pour un véhicule de T de se déplacer de a vers b. Chaque véhicule t de T est associé à une position $v \in V$ dans le graphe G, c'est-à-dire à un sommet de V.*

*Une **solution** d'un problème VRP est un ensemble de routes (une pour chaque élément de T) de sorte que chaque sommet de G soit visité au moins une fois et que tous les véhicules de T retournent à leur position initiale.*

Les solutions d'un VRP sont souvent classées au moyen d'une fonction de coût. On appelle la solution optimale d'un problème VRP la solution minimisant cette fonction de coût. Plusieurs fonctions de coût sont communément employées, par exemple : la durée de la plus longue route, l'écart de durée entre la plus longue et la plus courte des routes, etc.

Le problème de VRP a connu de nombreuses variations et extensions qui permettent de l'adapter à un grand nombre de scénarios. La liste suivante présente les variations les plus pertinentes au milieu de la chasse aux mines :

— **VRPTW** (VRP with Time Windows) [Sav84] où chaque sommet de G doit être atteint dans un certain intervalle de temps.
— **CVRP** (Capacited VRP) [LGW05] où le coût de la route associée à un véhicule t de T ne doit pas dépasser une certaine valeur de capacité c_t propre à t.

Une première approche notable a été proposée dans [Mah+15] où la zone de mission est représentée par un problème de VRPTW. L'approche traduit le problème par un ensemble de $waypoints$ à parcourir, tout en optimisant une fonction de coût et en respectant des contraintes temporelles sur la mission. Cette fonction de coût est une fonction hybride combinant différents objectifs pondérés par des poids définis manuellement. L'ensemble de contraintes est résolu grâce à un algorithme d'*optimisation par essaims particulaires* (ou PSO pour Particle Swarm Optimisation) [ZWJ15] couplé à un algorithme génétique (ou GA pour Genetic Algorithm) [Mit98]. Cette méthode a par la suite été étendue dans [MPY16] pour tenir compte des missions où la connaissance de l'environnement de mission est partielle. Enfin, une dernière version de ce système de planification de mission a été présentée dans [Mah+18]. Dans cette version, le problème de VRP est résolu à l'aide d'un algorithme d'optimisation biogéographique (ou *BBO* pour *Biogeograghy-Based Optimisation*) [Sim08]. Le principal intérêt de ces méthodes est de pouvoir trouver rapidement

une première solution au problème et de l'améliorer au fil des itérations. De plus ces méthodes d'optimisations sont théoriquement indépendantes de la taille et de la complexité des équations à résoudre, ce qui les rend intéressantes pour les problèmes de grandes tailles. Un autre exemple de ce type d'approche a été proposé dans [BST11] où la mission de la flotte est représentée par un problème de CVRPTW (Capacited VRP with Time Windows) et est résolue par un algorithme des fourmis couplé à une méta-heuristique.

Dans [Bia+09], la formulation VRP du problème permet aux appareils de la flotte de sélectionner le plus court chemin, en tenant compte de zones avec des contraintes particulières. Par exemple, certaines zones sont interdites ou dangereuses et donc à éviter autant que possible. Dans ce papier, le problème VRP est résolu à l'aide d'un algorithme de Dijkstra [Sni06] couplé à un algorithme génétique pour déterminer les vitesses des appareils.

Une autre approche notable a été proposée dans [Kar+14] où la mission représentée par un VRPTW est également planifiée en utilisant un algorithme de Dijkstra. Contrairement à l'approche précédente, un algorithme d'optimisation exacte est utilisé, ce qui permet de garantir certaines propriétés de la solution trouvée : plutôt que d'améliorer une première solution comme dans [Mah+15 ; MPY16], les approches combinatoires peuvent directement renvoyer une solution optimale. Cependant, ce type d'approche souffre d'un manque d'efficacité par rapport aux approches de type CSP, comme cela a été étudié dans [SSB12].

3.1.2 Approches par planification automatique

Dans le paragraphe précédent, nous avons présenté les systèmes de planification utilisant des approches par reformulation. Ces approches cherchent à exprimer le problème à résoudre en un type de problème connu, dans le but de pouvoir appliquer un solveur spécialisé sur ce type de problème. Les approches présentées dans ce paragraphe cherchent, à l'inverse, à résoudre un problème général de planification et sont ensuite appliquées au problème particulier de la chasse aux mines. On parle alors de *General Problem Solver* (ou GPS) [EN69 ; Nil09]. En d'autres termes, quand les approches par reformulation cherchent des analogies entre les contraintes de la chasse aux mines et celles de problèmes classiques, les approches de ce paragraphe cherchent des méthodes générales d'expression et de résolution à un problème général de planification.

La conséquence de ceci est que les systèmes utilisant la planification automatique sont plus facilement adaptables à plusieurs problèmes différents. En effet, pour un

système de planification automatique, une modification du problème à résoudre (modification de l'effet d'une action, ou ajout d'une capacité pour un véhicule) ne change pas le principe de résolution. Dans le cas des systèmes par reformulation, il faut trouver une façon de formuler la modification du problème dans l'ancienne formulation, ou en trouver une autre. L'un de principaux avantages de la planification automatique est donc sa grande adaptabilité quand il s'agit de planifier des missions dans des contextes et acteurs variés.

Pour cela, la planification automatique représente l'environnement comme un ensemble de propriétés. L'état dans le lequel se trouve l'environnement est caractérisé par la valeur de ces propriétés. Les agents du problème peuvent alors interagir avec l'environnement et modifier ses propriétés au moyen d'*actions*. La modélisation des propriétés dépend des *suppositions* qui sont faites sur cet environnement. Ces suppositions peuvent se présenter de la manière suivante [GNT04] :

— **Dynamique de l'environnement :** Certains systèmes peuvent supposer que l'environnement ne peut être modifié que par l'action (planifiée) des agents du problème. À l'inverse, d'autres systèmes autorisent l'environnement à se modifier indépendamment des agents.
— **Observabilité de l'environnement :** Suivant les systèmes, certaines propriétés de l'environnement peuvent être supposées comme connues à tout instant, ou alors possiblement observables par les agents, ou tout simplement inconnues et inaccessibles.
— **Certitude des connaissances :** Quand bien même ces propriétés seraient connues, une autre supposition possible est de considérer ces propriétés comme incertaines, c'est à dire associée à une certaine probabilité d'être véridique.
— **Temps et concurrence :** Les systèmes de planification peuvent considérer ou non que les actions des agents ont une durée, et qu'elles peuvent ou non s'exécuter en concurrence (c'est à dire simultanément).

Étant données l'*hypothèse du monde fermé* que nous avons fait dans la Partie 2.4.1, nous supposerons ici que l'environnement n'est pas dynamique et que les propriétés de l'environnement sont à la fois *connues en tout instant* et *certaines*. Enfin, la dimension temporelle des actions divise les systèmes de planification en deux catégories, les planificateurs temporels et non temporels, dont les différences seront discutées par la suite.

Pour former un problème de planification, la description du monde et des agents est couplées à une description du but du problème. Ce but peut être défini de trois façons différentes :

— **Comme un état objectif G à atteindre**. Dans ce cas, le but est d'atteindre un état S contenant toutes les propriétés de l'état objectif (c'est-à-dire $G \subseteq S$) à partir d'un état initial. On parle alors de **planification classique**.

— **Comme un ensemble de tâches à résoudre**, dans ce cas, le but est d'accomplir un ensemble de tâches à partir d'un état initial. Les tâches peuvent être soumises à des contraintes d'ordonnancement ou de simultanéité. On parle alors de **planification hiérarchique**.

— **Comme une combinaison de tâches à résoudre et d'un état objectif**, ce cas est simplement la combinaison des deux cas précédents. L'objectif est d'accomplir l'ensemble des tâches définies tout en atteignant l'état objectif. On parle alors de **planification hybride**.

De cette définition des problèmes de planification, il existe deux approches principales utilisées pour leur résolution. La première est de représenter le problème de planification comme un problème d'*allocation de tâches multi-robots* (ou *MRTA*). Le but est alors de répartir les tâches entre les différents appareils de la mission afin d'obtenir la répartition la plus efficace possible (au sens d'une fonction d'optimisation). Par la suite, on appellera ce type de planification la *planification par MRTA*. La seconde stratégie est de générer directement la séquence d'actions permettant de réaliser l'objectif de la mission. Ce type de planification ne dépend pas d'une structure en tâches ou sous-objectifs à distribuer, et elle sera par la suite nommée la *planification générale*.

Ces deux stratégies de représentation du problème impliquent des méthodes de résolution différentes qui seront détaillées par la suite.

Planification par MRTA

Comme énoncé précédemment, le problème d'allocation de tâches multi-robots (MRTA) consiste à distribuer un ensemble de tâches à un ensemble d'agents. Plus formellement, un problème MRTA peut être défini de la manière suivante :

Définition 5 *Un problème d'**allocation de tâches multi-robots (MRTA)** $P = (R, Q)$ est composé d'un ensemble de robots $R = \{r_1, r_2, \ldots, r_n\}$ et d'un ensemble de tâches $Q = \{t_1, t_2, \ldots, t_m\}$. Une solution de P est une allocation $A : Q \mapsto R$.*

De nombreuses allocations sont possibles pour un même problème de MRTA. La qualité d'une allocation est mesurée à l'aide d'une fonction de coût ou de récompense.

En ce sens, un problème de MRTA peut être vu comme un problème d'optimisation dans lequel il faut rechercher l'allocation optimisant ladite fonction.

Ceci dit, il faut distinguer une solution du problème MRTA et le plan de la mission globale. Une fois l'allocation réalisée, la manière dont ces tâches doivent être réalisées est encore à déterminer. La façon dont ces tâches sont planifiées peut varier d'un système à l'autre. De plus, les tâches peuvent être interdépendantes entre elles, ce qui impose des contraintes quant à la réalisation des tâches.

Aussi, il faut noter que de nombreuses variantes du problème de MRTA ont été définies. Une analyse des différentes variantes de ce problème est disponible dans [GM04].

Un premier exemple de système de planification par MRTA a été présenté dans [Sar07]. DEMiR-CF, le système proposé, utilise un système d'enchère pour attribuer les tâches aux appareils de la mission. Chaque appareil peut miser sur une tâche au moyen d'une fonction d'évaluation de son coût de réalisation de la tâche. Plus ce coût est élevé et plus la mise est faible. Puis une estimation de la durée globale de la mission est formulée à partir du résultat des enchères et la réalisation concrète des tâches est déterminée au moment de l'exécution. Le système d'enchère permet également aux appareils de s'échanger les tâches en cours de mission, en cas d'échec d'un des appareils.

Dans [SL10], une unité décisionnelle centralisée distribue des objectifs à accomplir aux appareils. L'utilisation d'un module central a pour effet de réduire le nombre de communications nécessaire à l'allocation des tâches. Les auteurs expliquent que cette approche est particulièrement adaptée aux milieux où les communications entre les appareils sont difficiles.

À l'inverse, le cas où les communications entre appareils sont parfaites a été étudié dans [SBD18a ; SBD18b]. Ce système propose une approche utilisant la *Logique Temporelle Linéaire* (*LTL*) pour décrire les dépendances temporelles entre les tâches. Les appareils utilisent des *processus de décision markovien* (ou MDP pour Markov Decision Process) pour établir la valeur des enchères. La particularité de l'approche est d'être capable d'apprendre des précédents problèmes résolus en améliorant les MDP à l'origine des enchères.

Une autre méthode d'allocation basée sur l'apprentissage a été proposée dans [Dai+20]. Cette fois-ci, l'apprentissage est réalisé au moyen de réseaux neuronaux couplés à un algorithme d'apprentissage par renforcement. Même si l'approche se révèle prometteuse pour les problèmes de plus petite taille, elle reste moins performante que les méthodes basées sur les systèmes d'enchères.

Le système présenté dans [Car+20] a la particularité d'allouer des objectifs à la place de tâches aux appareils. Cette approche permet d'utiliser la MRTA dans un cadre où la structure en tâche n'est pas disponible. L'approche utilise un algorithme de programmation linéaire pour déterminer le nombre d'appareils disponibles avant de répartir les objectifs de la mission aux appareils sélectionnés.

Un autre exemple d'utilisation de l'allocation par enchère a été proposé dans [Mil22]. Le système HaucTioN réalise son allocation au moyen d'un système d'enchères utilisant la structure hiérarchique des tâches à distribuer. Cette approche permet d'exploiter la structure des tâches pour déterminer l'interdépendance des tâches à réaliser. Cette approche est donc particulièrement intéressante lorsque les tâches sont complexes ou requièrent la coopération de plusieurs appareils. L'utilisation d'enchères pour la résolution de problèmes MRTA a vu de nombreuses contributions, une revue des approches publiées est disponible dans [QGL23].

Planificateurs généraux

Contrairement aux approches reposant sur la MRTA, certains planificateurs cherchent à résoudre la mission dans leur ensemble. Ces systèmes de planification sont en général moins efficaces que les systèmes MRTA. En effet, la MRTA permet une division du problème en sous-problèmes plus simples à résoudre. Cependant, une planification générale permet de considérer une mission dans son ensemble, et donc de garder un contrôle sur l'optimalité de la solution. Ainsi, les planificateurs généraux sont plus souvent utilisés dans des contextes *hors ligne*, où l'efficacité du planificateur n'est pas la priorité, mais où la qualité de la solution l'est.

Un premier exemple d'un tel planificateur a été proposé dans [Woo+05]. Ce système permet à l'utilisateur de paramétrer les spécificités de la mission, tant en ce qui concerne les objectifs à accomplir que l'ordre dans lequel il faut les accomplir. La planification des actions du véhicule, ainsi que la replanification au cours de la mission, se fait au moyen de réseaux de Petri [BC04].

Une autre approche de planification générale a été présentée dans [Cas+14]. Le sujet de la recherche est de démontrer l'efficacité d'un planificateur, POPF [Col+10], en l'appliquant au problème de la chasse aux mines. Ce planificateur réalise une recherche heuristique de la solution qui est indépendante du problème de la planification de mission d'AUV. L'intérêt de ce type de méthode est donc que l'efficacité de la résolution du problème est peu affectée par une modification du problème d'entrée.

Dans le même registre, le système EROPtus [Py+17] utilise un planificateur dénommé T-REX [RP12] qui utilise une méthode de division du problème en sous-problèmes indépendants afin de résoudre le problème global plus efficacement. La particularité du système est de permettre une planification *mixte*, qui permet à un opérateur d'introduire de nouveaux objectifs ou demander des modifications du plan solution au système.

La recherche heuristique de solution reste la méthode de résolution la plus populaire dans les planificateurs généraux. Le planificateur PUMMA décrit dans [MP16] utilise la logique temporelle linéaire pour décrire la mission à planifier. La particularité de l'approche est de coupler la recherche heuristique à la création de classe d'équivalence pour réduire la complexité du problème.

Enfin un dernier exemple plus récent de planificateur général a été publié [KBM22]. Ce planificateur exploite également une recherche heuristique pour rechercher sa solution, avec la particularité d'utiliser des heuristiques différentes en fonction du niveau de la planification : les actions de haut niveau ne sont pas planifiées par le même module que celles de plus bas niveau.

3.1.3 Commentaire intermédiaire

Jusqu'à présent, nous avons vu les systèmes utilisés pour la chasse aux mines et la planification de mission d'AUV. Nous avons présenté les avantages et les inconvénients de chaque méthode en fonction du besoin : quand les approches par reformulation sont généralement plus efficaces, mais aussi plus sensibles à une modification du problème d'entrée, les approches de la planification automatique permettent de planifier *indépendamment* du problème de planification. Au sein des approches de la planification automatique, on distingue deux catégories de systèmes. La première représente le problème de planification comme un un ensemble de tâches à allouer à des agents, la planification par MRTA. La seconde cherche directement une séquence d'actions permettant de réaliser l'ensemble des tâches et objectifs, la planification générale.

L'objectif de cette thèse est de proposer un planificateur capable d'*expressivité* et de *modularité*, en ce sens, les approches de la planification automatique répondent à cette attente. De plus, le planificateur désiré est un planificateur *hors ligne*, c'est-à-dire détaillant un plan avant la mission, à partir d'un environnement a priori. En ce sens, les approches qui nous semblent les plus aptes à répondre à ces problématiques sont les approches de la *planification automatique* et en particulier les *planificateurs généraux*.

Pour aller dans ce sens, la partie suivante met l'accent sur l'état des connaissances dans le domaine de la planification automatique sans plus se limiter au milieu de la chasse aux mines ou aux missions d'AUV.

3.2 Planification automatique : Un formalisme de planification général

Le domaine de la planification automatique englobe un grand nombre d'enjeux et de contraintes variés. En effet, c'est un domaine extrêmement large qui s'applique à de nombreux problèmes dont les spécificités peuvent grandement varier d'un cas à l'autre. Plus précisément, la *planification* est la discipline consistant à *déterminer à l'avance* les *actions à réaliser* dans le but d'*accomplir un objectif* défini a priori. La planification automatique est la discipline visant à résoudre les problèmes de planification de manière informatique. En d'autres termes, la planification automatique consiste à déterminer informatiquement les actions à entreprendre pour réaliser un objectif pré-établi, à partir d'une description initiale de l'environnement.

Tout système de planification ne peut fonctionner que s'il possède à la fois une connaissance a priori de *l'environnement* dans lequel la planification a lieu, mais aussi une connaissance des *effets des actions* qu'il peut planifier. Ce sont ces connaissances qui permettent la prise de décision ainsi que la création d'un plan. En planification, la description des actions réalisables ainsi que la description de leurs effets sur l'environnement sont dénommées un *domaine de planification*. La description initiale de l'environnement ainsi que l'objectif à réaliser forment une *instance* de problème. Nous rappelons que toutes les propriétés de l'environnement sont supposées être connues à tout instant, ainsi que certaines. De plus, l'environnement est lui-même supposé non-dynamique (voir Parties 2.4.1 et 3.1.2).

Dans l'exemple de la chasse aux mines, le domaine de planification décrit les actions réalisables par les appareils de la flotte. Par exemple, couvrir une zone avec un drone de détection permet de produire une image SAS. Ou alors, certains appareils peuvent réaliser des déplacements et ainsi modifier leur position. L'instance de problème, quant à elle, décrit la zone de mission à déminer ainsi que les contraintes environnementales et/ou opérationnelles associées à cette zone.

Il faut noter qu'un domaine de planification peut être utilisé avec plusieurs instances. En effet, le domaine permet de définir les capacités de chaque agent en fonction de la situation dans laquelle il se trouve. Une instance décrit quant à elle une situation

particulière dans laquelle un objectif doit être atteint. Le principe du domaine est de décrire les capacités de chaque appareil pour ensuite utiliser ces appareils dans des situations spécifiquement décrites par des instances de problème.

Planification dépendante du domaine

Une première approche naturelle de la planification est de développer des systèmes spécifiquement conçus pour résoudre des problèmes issus d'un même domaine de planification. L'objectif de cette approche est d'utiliser des connaissances spécifiques au domaine de planification pour générer efficacement des plans solutions. On dit de ce type de planification qu'il est *dépendant du domaine* de planification.

Les approches dépendantes du domaine sont des plus légitimes. Elles sont généralement les approches les plus efficientes, autant d'un point de vue du temps dépensé pour trouver la solution que d'un point de vue de la qualité du plan retourné.

L'exemple le plus explicite de planification dépendante du domaine est certainement l'exemple d'AlphaGo [Sil+16]. Ce système est capable de planifier une séquence de coup au jeu de go, et est capable de le faire si efficacement qu'il a réussi à battre les meilleurs humains. Cependant, AlphaGo est uniquement pertinent dans le cadre bien spécifique du jeu de go. Il ne permet pas de jouer à d'autres jeux comme Othello ou les échecs, ni même au jeu de go sur un plateau de taille différente. En ce sens, AlphaGo représente les avantages et les inconvénients de la planification dépendante du domaine. Elle permet de planifier de manière particulièrement efficace, mais dans un cadre spécifiquement défini. Toute modification de ce cadre peut donc remettre en cause l'efficacité du planificateur. L'un des principaux enjeux du domaine de l'intelligence artificielle est d'ailleurs de produire des intelligences artificielles dites *générales*, c'est-à-dire non limitées à un cadre d'application particulier [Bar+17 ; Sze20 ; SS19].

Finalement, on peut résumer les principaux inconvénients de la planification dépendante du domaine en deux points [GNT04] :

— Elle se concentre sur les contraintes spécifiques à son domaine de planification et ne permet pas de faire progresser la planification dans le cas général.
— Chaque modification du domaine implique une réadaptation du planificateur en retour. Ce processus peut se montrer coûteux.

Pour toutes ces raisons, la suite de cette thèse se concentrera sur la planification dite *indépendante du domaine*, c'est à dire qui peut s'appliquer à n'importe quel domaine

de planification indifféremment, du moment où il peut être exprimé dans un langage de planification.

3.2.1 Décrire un problème de planification

Si l'on souhaite parler de planification automatique, la première étape est d'être capable de décrire le problème à résoudre. À ce propos, plusieurs langages de planification ont été proposés. Ces langages diffèrent en expressivité, c'est-à-dire dans leur capacité à représenter des problèmes de planification. Comme cela a été évoqué auparavant, il existe plusieurs manières de définir un objectif. Historiquement, les langages de planification se sont concentrés sur la *planification classique*, où l'objectif est défini comme un état objectif à atteindre. Les langages pour la *planification hiérarchique* et *hybride*, où l'objectif est respectivement un ensemble de tâches à réaliser ou une combinaison de tâches et d'un état objectif, ont été proposés dans un second temps. Ainsi, nous commençons par présenter les langages de la planification classique, puis de la planification hiérarchique et hybride.

Langages de planification classique

Pour commencer, la première formalisation d'un problème de planification classique a été proposée par le planificateur STRIPS [FN71]. Le principe de ce langage est de définir un état du problème comme un ensemble de propriétés booléennes. Les agents de la mission peuvent modifier l'état du problème au moyen d'*actions*. Ces actions sont définies par deux ensembles de propriétés booléennes, un ensemble de *préconditions* (l'ensemble des propriétés qui doivent être vérifiées pour appliquer l'action), ainsi qu'un ensemble d'*effets* (l'ensemble des propriétés modifiées par l'application de l'action). L'application d'une action à un état est représentée par une *fonction de transition*, elle associe à un couple état/action l'état résultant de l'application de l'action. Cette formulation est la plus communément utilisée pour représenter un problème de planification classique :

Définition 6 *Un problème de* ***planification STRIPS*** $P = (L, A, I, G)$ *est un quadruplet où :*

— L *est un ensemble fini de propositions logiques booléennes.* L *contient l'ensemble des propriétés intervenant dans le problème. On appellera un état tout sous-ensemble de* L*.*

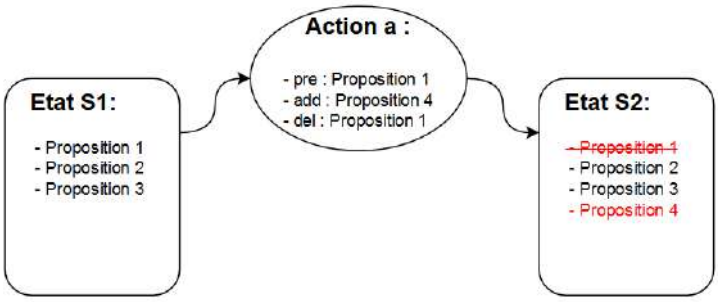

Figure 3.2. : Application d'une action a à un état $S1$. a est applicable à $S1$ car $pre(a) \subseteq S1$ et $S2 = \gamma(S1, a) = (S1 - del(a)) \cup add(a)$

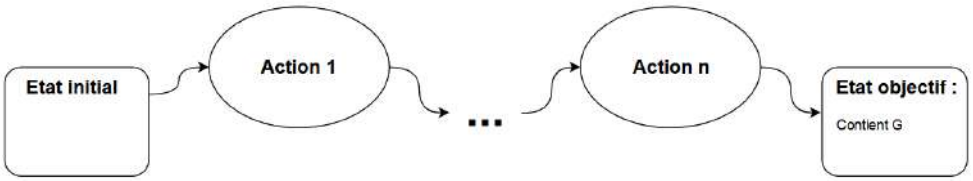

Figure 3.3. : Représentation d'un plan solution, un plan solution est valide si les actions le composant sont successivement applicables aux états résultant de l'état initial et si l'état final contient l'état objectif G du problème. En d'autres termes si $\gamma(I, \langle a_1, ..., a_2 \rangle) \supseteq G$.

— *A est un ensemble fini d'actions. Une action $a = (pre(a), add(a), del(a))$ est un triplet de sous-ensembles de L où $pre(a)$ représente l'ensemble des propositions à vérifier pour que l'action soit applicable à un état, $add(a)$ représente les propositions qui seront ajoutées à l'état par l'application de l'action et $del(a)$ celles qui en seront retirées. On notera $\gamma : L \times A \mapsto L$ la fonction de transition du problème, et $\gamma(s, a)$ dénotera l'état résultant de l'application de l'action a à un état s, et tel que $\gamma(s, a) = (s - del(a)) \cup add(a))$. Un exemple d'application d'actions est représenté dans la Figure 3.2*

— *$I \subseteq L$ est un ensemble de propositions logiques contenu dans L représentant l'état initial du problème et $G \subseteq L$ est un autre ensemble de propositions représentant l'état objectif du problème.*

On appelle un plan solution de P toute séquence d'action $\pi = \langle a_1, ..., a_n \rangle$ qui, appliquée à l'état initial, résulte en l'état final. En d'autres termes, π est un plan solution si $G \subseteq \gamma(I, \pi) = \gamma(\gamma(I, a_1), \langle a_2, ..., a_n \rangle)$. Il faut noter que tout état contenant G est considéré comme un état objectif, plusieurs états objectifs sont donc possibles. Une représentation d'un plan solution est affichée dans la Figure 3.3.

Le premier langage de planification classique est proposé par PDDL (pour *Planning Domain Definition Language*)[How+98]. Ce langage est basé sur la formulation

STRIPS des problèmes, et offre les mots-clés nécessaires à l'expression des problèmes de planification. Il présente cependant de nombreuses limites. En particulier, il considère les actions des agents comme instantanées, ce qui ne permet pas de modéliser des contraintes temporelles ou encore la concurrence entre les actions.

À ce propos, de nouvelles versions de PDDL ont été proposées. Elles sont dénommées PDDL2.1 [FL03] et PDDL+ [FL02]. Ces deux versions de PDDL rajoutent chacune un niveau d'expressivité au langage initial PDDL. Tout d'abord, PDDL2.1 ajoute la prise en compte des *actions duratives*, qui se déroulent sur un intervalle de temps plutôt que d'être instantanées. Puis PDDL2.2 permet la modélisation de variables continues. Les propriétés de l'environnement et des agents ne se limitent donc plus aux variables booléennes. Finalement, PDDL3 [GL05] a permis la définition de préférences sur l'état objectif à atteindre ainsi que de nouvelles contraintes sur la façon d'atteindre l'état objectif.

Un autre langage de planification classique a été proposé par le planificateur de la NASA, EUROPA [Bed+05]. Le langage associé, NDDL, n'utilise pas de notion d'état, d'action ou d'état objectif. Au contraire, le langage définit un ensemble d'activités qui peuvent être réalisées en affectant des ressources à cette activité pendant un certain intervalle temporel. Les modèles écrits en NDDL sont cependant plus verbeux et plus difficiles à utiliser que ceux de PDDL.

Avec le temps donc, PDDL et ses extensions se sont imposés comme le langage de facto dans les compétitions de planification. En particulier, la compétition annuelle IPC (*International Planning Competiton*) [Val+15 ; BHB21] utilise PDDL depuis 2006 [Ger+09]. Ce langage est le plus communément utilisé dans le milieu académique et de nombreux planificateurs sont compatibles avec les problèmes exprimés en PDDL et ses extensions. De plus, de nombreux problèmes de référence ont été développés pour ces compétitions, ce qui facilite les tests et la comparaison entre les approches de résolution.

Langages de planification hiérarchique

La seconde grande approche pour formuler un problème de planification automatique est la planification hiérarchique. Elle ressemble à la planification classique en ce qu'elle utilise des propositions logiques pour représenter le monde et des actions pour décrire les transitions d'état. Toutefois, au lieu de définir un objectif comme un état final à atteindre, la planification hiérarchique définit un réseau hiérarchisé de tâches, appelé Hierarchical Task Network (HTN) [GNT04]. Ces tâches peuvent être primitives ou abstraites. Une tâche primitive est associée à une action au sens de la

planification STRIPS. Une tâche primitive est résolue lorsque l'action correspondante est appliquée à l'état courant. Une tâche abstraite, quant à elle, est associée à un ensemble de méthodes décrivant comment la décomposer en sous-tâches, elles-même primitives ou abstraites.

La résolution d'un problème hiérarchique s'effectue donc en deux phases. D'abord, le réseau initial de tâches est décomposé en tâches primitives à l'aide de méthodes. Ensuite, l'objectif est de trouver une suite d'actions STRIPS qui résolve l'ensemble des tâches (primitives) du réseau, tout en respectant la hiérarchie de celui-ci.

Plus formellement, un problème de planification hiérarchique se définit de la manière suivante [GB11] :

Définition 7 *Un problème de* ***planification hiérarchique*** $P = (L, \mathcal{T}, A, M, I, tn)$ *est un quintuplet composé de :*

- L *un ensemble fini de propositions logiques booléennes.*
- $\mathcal{T}$ *est un ensemble fini de tâches. Ces tâches peuvent être de deux types, primitives ou abstraites.*
- A *est un ensemble fini d'actions au sens de la planification STRIPS (voir le paragraphe 6).*
- M *est un ensemble fini de méthodes. Une méthode* $m \in M$ *est un triplet* $m = (task(m), pre(m), tn(m))$ *où* $task(m)$ *est la tâche (nécessairement abstraite) de* $\mathcal{T}$ *décomposée par* m*,* $pre(m)$ *est un ensemble de préconditions de* m *(au sens d'une précondition STRIPS) et* $tn(m)$ *représente le HTN qui remplacera* $task(m)$*.*
- tn *représente le réseau initial de tâches du problème. Un réseau de tâches* $tn = (T, \prec, \alpha)$ *est lui-même composé (1) d'un ensemble fini de tâches* $T = \{t_1, t_2, \ldots, t_k\}$ *issues de* $\mathcal{T}$ *(*$\forall i, t_i \in \mathcal{T}$*), (2) d'une relation d'ordre partielle* $\prec$ *sur les tâches de* T *et d'une fonction* $\alpha : T \mapsto \mathcal{T}$ *associant chaque symbole de* T *à une tâche de* $\mathcal{T}$*. Si* $t_1, t_2 \in T$*, on dit que* t_1 *précède* t_2 *ssi* $t_1 \prec t_2$*. Il faut noter que* $\prec$ *est transitive : si* $t_1, t_2, t_3 \in T$*, et si* $t_1 \prec t_2$ *et* $t_2 \prec t_3$ *alors* $t_1 \prec t_3$*.*

Un réseau de tâches $tn = (T, \prec, \alpha)$ *peut être décomposé par une méthode* $m = (task(m), pre(m), tn(m))$ *où* $tn(m) = (T_m, \prec_m, \alpha_m)$ *si* $\exists t \in T, t = task(m)$*. Le réseau de tâches résultant est un réseau* $tn' = (T', \prec', \alpha')$ *où :*

$$
\begin{aligned}
T' &= (T \setminus \{t\}) \cup T_m \\
\prec' &= \prec \cup \prec_m \\
&\quad \cup \{(t_1, t_2) \in T_m \times (T \setminus \{t\}) \mid (t, t_2) \in \prec\} \\
&\quad \cup \{(t_1, t_2) \in (T \setminus \{t\}) \times T_m \mid (t_1, t) \in \prec\} \\
\alpha' &= \{(t', \alpha(t')) \mid t' \in T \setminus \{t\}\} \cup \alpha_m
\end{aligned}
$$

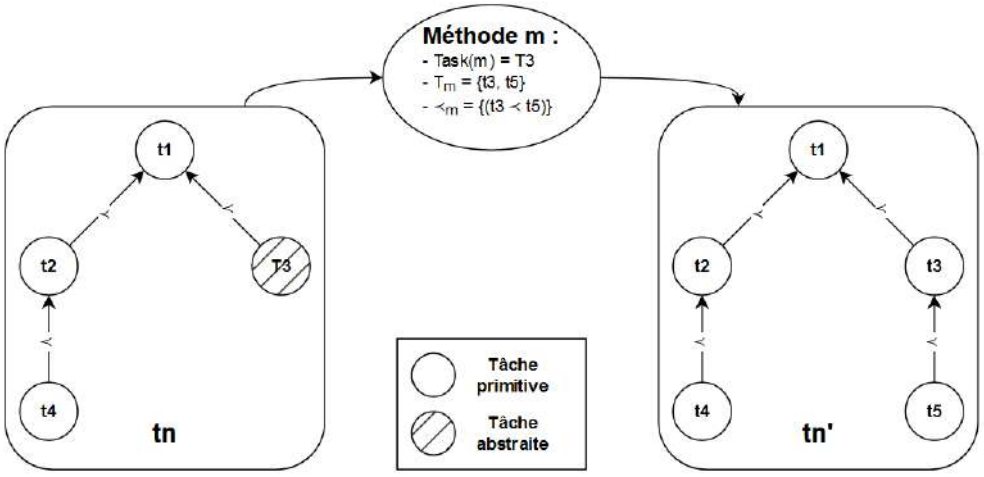

Figure 3.4. : Décomposition d'un réseau de tâches au moyen d'une méthode. La tâche décomposée est remplacée par le réseau de tâches de la méthode.

En d'autres termes, les sous-tâches issues de la décomposition de t par m sont introduites dans le réseau de tâches à la place de t. Les contraintes qui impliquaient t sont reportées sur les sous-tâches introduites.

Un exemple de décomposition d'un réseau de tâches est représenté dans la Figure 3.4.

Un plan $\pi = \langle a_1, ..., a_n \rangle$ est une solution de $P = (L, \mathcal{T}, A, M, I, tn)$ s'il existe un réseau de tâches $tn_f = (T_f, \prec_f, \alpha_f)$ issu de décompositions successives de tn ainsi qu'une bijection $\sigma : T_f \mapsto \{a_1, \ldots, a_n\}$ telle que $\forall t \in T_f, \sigma(t)$ résout t. Enfin, si $t_1, t_2 \in T_f$ telles que $t_1 \prec_f t_2$ alors $\sigma(t_1) \prec \sigma(t_2)$. La Figure 3.5 représente un exemple d'un tel plan.

Enfin, un problème de planification hiérarchique peut être défini avec un état objectif à atteindre en supplément du réseau de tâches. On dit alors que $P = (L, \mathcal{T}, A, M, I, tn, G)$ est un problème de ***planification hybride****. L'état objectif G est défini à la manière d'un problème STRIPS (voir Définition 6). Un plan solution d'un problème hybride est donc un plan solution du problème hiérarchique dont l'état final contient G, un exemple est représenté dans la Figure 3.6.*

La planification hiérarchique présente de nombreuses propriétés qui la rendent intéressante à plusieurs niveaux. Tout d'abord, la structure en tâches partiellement ordonnées est plus facilement utilisable par un opérateur humain. En effet, définir un ensemble de tâches à réaliser est plus naturel que de déterminer l'ensemble des propriétés booléennes nécessaires à l'accomplissement de la mission. En plus de cela, la création du plan solution via la décomposition du réseau de tâches initial

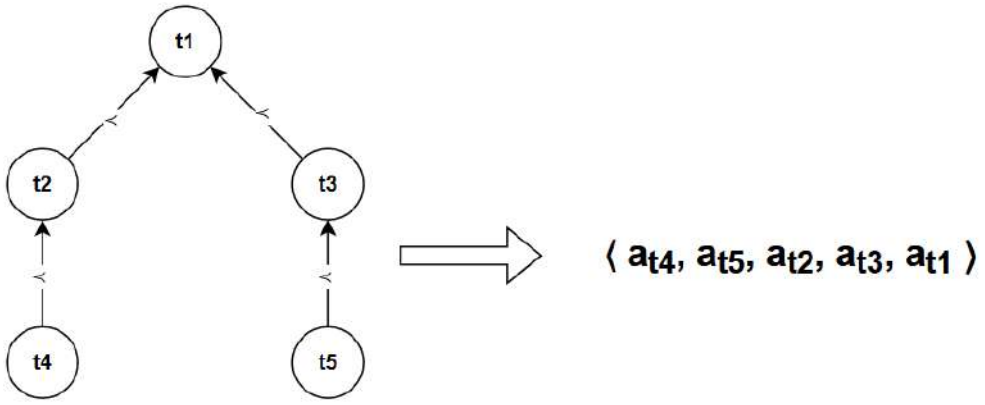

Figure 3.5. : Exemple de plan solution d'un problème hiérarchique, la suite d'actions doit résoudre un réseau de tâches issu de la décomposition du réseau initial.

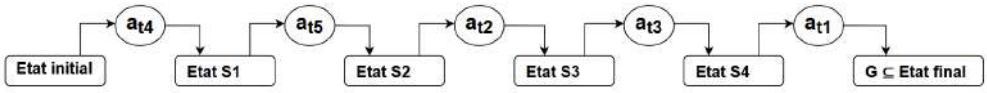

Figure 3.6. : Une solution d'un problème hybride doit répondre au problème hiérarchique tout en atteignant l'état objectif du problème.

permet de *justifier* la présence de chaque action. Chaque action est ainsi vue comme une étape permettant la réalisation d'une tâche plus large, elle-même faisant partie d'une mission plus globale. Cet aspect est particulièrement important quand la planification fait intervenir des appareils critiques, comme dans le cas de la chasse aux mines.

Pour toutes ces raisons, les premiers essais de formalisation de langages de problèmes hiérarchiques ont été fais dans le cadre d'application concrète. La première proposition notable a été faite par AML [Fuk+97]. Ce langage a été développé par la NASA dans le but de réaliser de la planification robotique, à l'image de NDDL [FJ03]. À l'instar de NDDL, AML reprend la notion d'*activités* à réaliser qui sont, cette fois, organisées par un ordre partiel. Contrairement à NDDL, AML réalise ses activités en les décomposant à la manière d'une décomposition HTN. Les activités réservent des ressources pendant un certain intervalle de temps, et pour AML, trouver un plan solution signifie ordonner ces intervalles de temps de sorte à n'avoir aucun conflit de ressources.

Une version plus aboutie de langage hiérarchique a été présentée avec ANML [SFC08]. Ce langage développé par la NASA a été pensé comme une alternative au langage PDDL et ses extensions PDDL+ et PDDL2.1 (voir le paragraphe 3.2.1) tout en incorporant les éléments propres à la planification hiérarchique. L'approche a

permis une première formulation complète de la planification hiérarchique couplée aux avancées des langages issus de la planification classique. Cependant, l'expression des problèmes au travers d'ANML reste fortement éloignée de celle de PDDL, ce qui pose des problèmes d'adaptabilité pour les planificateurs classiques existants.

Une première version d'extension hiérarchique de PDDL a été présentée sous la forme de HPDDL [AKN09]. Ce langage permet une expression de problème hiérarchique avec un même formalisme (augmenté) que PDDL. Il présente cependant des limites. En particulier, HPDDL ne supporte pas l'ordre partiel des tâches dans un réseau. Ce qui présente des limites dans l'expressivité des problèmes formulés.

Les limites des langages de planification hiérarchique ont poussé les planificateurs à développer leurs propres langages, spécifiquement définis pour leurs besoins, comme dans le cas du planificateur HTAP [DLA15] qui définit un langage hiérarchique proche de celui proposé par ANML [SFC08]. HTAP présente cependant les mêmes défauts que HPDDL, à savoir que les décompositions des tâches abstraites ne peuvent se faire que par une séquence totalement ordonnée de tâches.

Finalement, un langage ayant pour ambition d'unifier le formalisme hiérarchique avec le langage de référence PDDL a été publié dans [Höl+20a]. HDDL est une extension de PDDL permettant d'exprimer l'intégralité du formalisme hiérarchique. La première version se limite aux problèmes donc les actions sont non temporelles (c'est-à-dire instantanées), ce qui a ensuite été résolu par la seconde version HDDL2.1 [Pel+23] qui permet justement la prise en compte des actions ainsi que des contraintes temporelles. Le langage HDDL est utilisé dans la compétition internationale IPC comme langage de référence. Une fois encore, ceci signifie que bon nombre de problèmes de tests ont été créés, ce qui est un intérêt majeur pour le test et la comparaison de planificateurs. HDDL est, à ce jour et à notre connaissance, la version la plus complète d'un langage de planification HTN.

3.2.2 Exemple de problème de planification

Dans le paragraphe précédent, nous avons passé en revue les langages de planification existants. Deux de ces langages, PDDL [How+98] et HDDL [Höl+20a] (ainsi que leurs extensions) servent de référence dans le milieu académique grâce à leur expressivité et la simplicité de leur syntaxe. L'objet de cette partie est de donner un exemple de formulation d'un problème de planification de chasse aux mines à l'aide de ces deux langages.

Figure 3.7. : Situation initiale du problème, l'AUV dans la zone de départ doit réaliser l'image SAS des trois sous-zones de mission.

Nous prenons le cas de deux AUV ayant pour mission de réaliser l'image SAS (Synthetic Aperture Sonar) de trois sous-zones de mission nommées $Z1$, $Z2$ et $Z3$. Les AUV sont initialement dans une quatrième zone $Z0$ qui leur sert de position initiale. Un schéma de la situation est représenté dans la Figure 3.7.

La représentation du problème de planification passe d'abord par la description de l'environnement dans lequel se déroule la mission ainsi que des agents qui pourront interagir à l'intérieur de celui-ci. Nous verrons dans un premier temps comment PDDL définit ceci dans le cadre de la planification classique, puis nous verrons comment HDDL le définit en planification hiérarchique.

Planification classique : formulation PDDL Le langage PDDL utilise la formulation de problème STRIPS ; il représente donc l'environnement et les agents du problème à l'aide de propriétés logiques. Dans le but de rendre l'utilisation de ces propriétés logiques aisée, PDDL utilise une représentation *lifted* du problème. Pour cela, les éléments du problème (les éléments de l'environnement et les agents) sont représentés en PDDL par des objets structurés en types et sous-types. La structure associée à notre exemple est représentée sur la Figure 3.8 : il y a deux types d'objet, des véhicules et des zones de mission. Les zones de mission étant elles-mêmes subdivisées en deux sous-types : les zones de départ et les zones de scan. Les véhicules ont ici un unique sous-type, les AUV. L'équivalent PDDL de la déclaration de ces types est affiché sur le Listing 3.1

Ensuite, PDDL permet de définir les propriétés de tous les appareils d'un même type (ou sous-type) au moyen de *prédicats*. Ces prédicats prennent un ou plusieurs objets en paramètres et renvoient la valeur de la proposition logique associée. Par exemple, la position des AUV au cours de la mission est modélisée au moyen du prédicat pos ayant pour paramètres un objet de type AUV ainsi qu'un objet de type $Zone\_Mission$. Si $auv1$ est un AUV et $z1$ une $Zone\_Mission$ alors $pos(auv1, Z1) = true$ modélise le fait que $auv1$ est dans la zone $Z1$. Dans cet

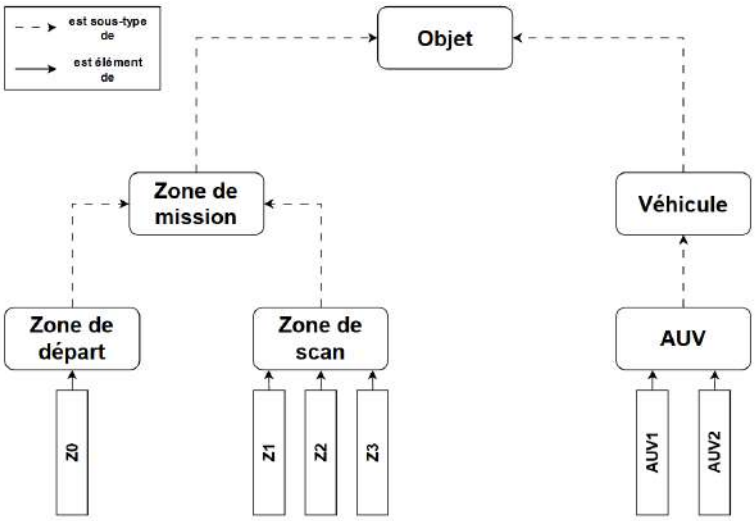

Figure 3.8. : Représentation de la structure des objets de la mission exemple.

```
(define (domain MCMMission) ; Nom du domaine
  (:requirements ; Modules PDDL requis
      :strips
    :typing
    :negative-preconditions
  )

(:types Zone_Mission Vehicule
  Zone_Depart Zone_Scan - Zone_Mission
  AUV - Vehicules)
```

Listing 3.1 : Définition PDDL des types de la mission.

exemple, nous considérerons deux prédicats : pos ainsi qu'un deuxième prédicat, $scanned$ prenant en paramètre un objet de type $Zone\_Scan$ et caractérisant le fait que la zone ait déjà été scannée. La définition PDDL de ces prédicats est affichée sur le Listing 3.2.

```
(:predicates
  (pos ?auv - AUV ?z - Zone_Mission)
  (scanned ?z - Zone_Scan)
)
```

Listing 3.2 : Déclaration des prédicats du problème.

Enfin la valeur des prédicats peut être modifiée au moyen d'opérateurs appelés *actions*. Ces actions sont considérées comme des fonctions prenant des objets en paramètres. Elles sont composées d'un ensemble de propositions (exprimées par des prédicats) à vérifier pour pouvoir être capable d'appliquer l'action, ainsi que d'un ensemble d'effets, c'est-à-dire de modifications de valeurs pour les prédicats.

Dans notre exemple on considérera deux actions, une action $move$ modélisant un déplacement d'AUV et une action $scan$ modélisant la réalisation d'une image SAS d'une zone. La représentation PDDL de ces deux actions est affichée sur le Listing 3.3. L'action $move$ a trois objets en paramètres, le premier $?auv$ et de type AUV et les deux autres $?from$ $?to$ sont de types $Zone\_Mission$. Pour pouvoir appliquer l'action à un état, le prédicat *(pos ?auv ?from)* doit être vrai. L'application de l'action modifiera la valeur du prédicat *(pos ?auv ?from)* et rendra vrai *(pos ?auv ?to)*. Ainsi, l'action $move$ modélise un changement de position de l'AUV.

```
(:action move
  :parameters (
    ?auv - AUV
    ?from ?to - Zone_Mission
  )
  :precondition (and
    (pos ?auv ?from)
  )
  :effect (and
    (not (pos ?auv ?from))
    (pos ?auv ?to)
  )
)

(:action scan
  :parameters (
    ?auv - AUV
    ?to_scan - Zone_Scan
  )
  :precondition (and
    (pos ?auv ?to_scan)
  )
  :effect (and
    (scanned ?to_scan)
  )
)
```

Listing 3.3 : Déclaration des actions du problème.

Jusqu'à présent, nous avons défini la structure des types d'objet du problème, en définissant les propriétés de ces types ainsi que les actions permettant de changer la valeur de ces propriétés. Tous ces éléments forment un *domaine de planification*. Le *domaine* est défini de manière indépendante du problème particulier que l'on souhaite résoudre. En d'autres termes, la définition du *domaine* ne varie pas en fonction du nombre de zones ou d'AUV dans le problème.

La déclaration concrète des éléments particuliers du problème se fait dans un second fichier appelé *fichier d'instance*. Dans celui-ci, sont déclarés (1) les objets du problème (avec leurs types correspondants) (2) l'état initial du problème (l'ensemble des propositions vraies à l'initialisation) ainsi que (3) l'état objectif à atteindre

(l'ensemble des propriétés devant être vérifiées dans l'état final). La modélisation PDDL du fichier d'instance de notre exemple est représentée sur le Listing 3.4. Dans l'état initial, la position initiale des deux AUV est déclarée à l'aide du prédicat pos. L'objectif est défini avec le second prédicat $scanned$: l'objectif de la mission est d'atteindre un état où toutes les zones ont été scannées.

```
(define
  (problem MissionTest)
  (:domain MCMMission)
  (:objects
    auv1 auv2 - AUV
    Z0 - Zone_Depart
    Z1 Z2 Z3 - Zone_Scan
  )
  (:init
    (pos auv1 Z0)
    (pos auv2 Z0)
        )
    (:goal (and
    (scanned Z1)
    (scanned Z2)
    (scanned Z3)
    )
  )
)
```

Listing 3.4 : Fichier d'instance associé au problème exemple (voir Figure 3.7).

Planification hiérarchique : formulation HDDL La formulation HDDL présente de nombreuses similarités avec la formulation classique. Tout d'abord la déclaration des types, des prédicats ainsi que des actions se fait de manière identique. Cependant, la planification hiérarchique définit également un ensemble de *tâches abstraites*, qui peuvent être décomposées en tâches plus simples au moyen de *méthodes*. Dans HDDL, à l'instar des actions PDDL, la déclaration des tâches abstraites se fait de manière *lifted* également. Une tâche abstraite peut donc avoir plusieurs paramètres qui sont des objets du problème. Dans notre exemple, on considère une unique tâche abstraite get_image ayant un unique objet de type $Zone\_Scan$ comme paramètre. La déclaration HDDL de cette tâche est représentée sur le Listing 3.5.

Une tâche abstraite peut avoir plusieurs réalisations possibles. Chaque façon de réaliser une tâche abstraite est définie par une *méthode*. Ces méthodes définissent deux choses : (1) quelles tâches seront introduites par la décomposition de la tâche abstraite et (2) dans quel ordre ces tâches seront introduites. Dans notre exemple, la tâche abstraite get_image peut être réalisée de deux façons différentes, en fonction de la position de l'AUV par rapport à la zone. Les deux méthodes associées sont

```
(define (domain MCMMission) ; Nom du domaine
  (:requirements
      :strips
    :typing
    :hierarchy ; Mot clef HDDL
    :negative-preconditions
  )

(:types Zone_Mission Vehicule
  Zone_Depart Zone_Scan - Zone_Mission
  AUV - Vehicules)

(:predicates
  (pos ?auv - AUV ?z - Zone_Mission)
  (scanned ?z - Zone_Scan)
)

(:task get_image ; Definition de la tache abstraite
  :parameters (?Z - Zone_Scan)
)
```

Listing 3.5 : Déclaration des prédicats du problème.

représentées sur le Listing 3.6. Les méthodes prennent des objets en paramètres et décomposent la tâche déclarée après le mot clef *:task*. La méthode est applicable si les préconditions sont vérifiées et introduit les sous-tâches déclarées après *:subtasks* dans l'ordre défini dans *:ordering*.

```
(:method get_image_move
    :parameters (?auv - AUV ?from - Zone_Mission ?to - Zone_Scan)
    :task (get_image ?to)
    :precondition (and
        (not (pos ?auv ?to))
        (pos ?auv ?from)
      )
    :subtasks (and
        (t1 (move ?auv ?from ?to))
        (t2 (scan ?auv ?to))
      )
    :ordering (and
      (< t1 t2)
    )
)
```

Listing 3.6 : Déclaration des méthodes du problème.

Enfin, et à l'instar de PDDL, HDDL définit l'état initial ainsi que l'objectif de la mission dans un fichier séparé. L'objectif de la mission est défini de manière hiérarchique, à l'aide d'un réseau de tâches, mais peut également être couplé à un état objectif à atteindre afin de formuler un problème de planification hybride. Le Listing 3.7 représente l'objectif de la mission exemple. Les objets de la mission sont définis de

la même manière que pour la planification classique, mais l'objectif est composé de trois tâches à réaliser, chacune permettant de réaliser l'image SAS d'une zone.

```
(define
  (problem MissionTest)
  (:domain MCMMission)
  (:objects
    auv1 auv2 - AUV
    Z0 - Zone_Depart
    Z1 Z2 Z3 - Zone_Scan
  )
  (:init
    (pos auv1 Z0)
    (pos auv2 Z0)
        )
    (:htn
      :tasks (and
        (t1 (get_image Z1))
        (t2 (get_image Z2))
        (t3 (get_image Z3))
      )
    :ordering ( )
  )
)
```

Listing 3.7 : Fichier d'instance associée au problème exemple (voir Figure 3.7).

Il faut noter que dans cet exemple, le mot clef *:ordering* est laissé vide. Cependant il est possible d'y spécifier des contraintes d'ordre sur les tâches de la mission. Par exemple, si l'on souhaite spécifier que l'image de la zone $Z1$ doit être réalisée avant celle de la zone $Z2$ et $Z3$, on peut ajouter un ordonnancement affiché sur le Listing 3.8. Cela illustre l'avantage de la planification hiérarchique dans l'expression de missions complexes : ce type de contrainte d'ordre serait possible à exprimer dans un contexte de planification classique, mais cela serait beaucoup plus laborieux et demanderait la définition de prédicats ou de contraintes supplémentaires pour encoder ces relations d'ordres entre les tâches d'exploration.

```
:ordering (and
  (< t1 t2)
  (< t1 t3)
)
```

Listing 3.8 : Fichier d'instance associée au problème exemple (voir Figure 3.7).

Formulation d'un problème temporel Dans les paragraphes précédents, nous avons exprimé un exemple de problème de planification dans sa formulation classique et hiérarchique. Cependant, dans ces deux formulations, les actions étaient considérées

comme instantanées. Cette supposition est une forte limitation quand il s'agit de formuler des problèmes complexes, en particulier lorsque la synchronisation entre les appareils est un enjeu majeur du problème. Pour formuler ces problèmes, les langages PDDL et HDDL présentent des extensions temporelles où des actions duratives sont définies. La définition communément adoptée pour les actions temporelles a été fournie par PDDL2.1 [FL03]. Contrairement aux actions instantanées qui ont simplement un ensemble de préconditions à vérifier, une action temporelle a trois ensembles de préconditions, qui doivent être vérifiés à différents moments de l'exécution de l'action :

1. Un ensemble de conditions doit être vérifié *avant* l'exécution de l'action, on parle de conditions *at Start*
2. Un ensemble doit être vérifié *pendant* son exécution, on parle de conditions *invariantes*.
3. Un ensemble de conditions requis pour *terminer* l'action, on parle de conditions *at End*.

Une action temporelle applique également des effets à deux moments différents, d'abord au démarrage de l'action, puis au moment de la terminer. Plus simplement, on peut voir une action temporelle comme deux actions instantanées (non temporelles), espacées par une certaine durée et dont un ensemble de propriétés doit être vérifié entre ces deux actions. Plus formellement, une action temporelle peut être définie ainsi :

Définition 8 *Une action temporelle* $a = (name(a), start(a), end(a), inv(a), d)$ *est un quintuplet dans lequel :*

— $nom(a)$ *est le nom de l'action* a*.*

— $start(a) = (pre_{atStart}(a), del_{atStart}(a), add_{atStart}(a))$ *est l'action instantanée correspondant au démarrage de* a*.*

— $end(a) = (pre_{atEnd}(a), del_{atEnd}(a), add_{atEnd}(a))$ *est l'action instantanée correspondant à la fin de* a*.*

— $inv(a)$ *est l'ensemble des propriétés qui doivent être vérifiées tout au long de l'exécution de* a*.*

L'introduction de la dimension temporelle implique également de définir précisément les intervalles sur lesquels les différentes conditions doivent être vérifiées. Si l'action a *s'exécute de l'instant* t_1 *à l'instant* t_2 *(et donc* $t_2 - t_1 = d$*), alors :*

— $inv(a)$ *doit être vérifié sur l'intervalle de temps ouvert* $]t_1, t_2[$.

— $pre_{atStart}(a)$ *doit être vérifié sur un intervalle* $]t', t_1[$ *avec* $t' < t_1$

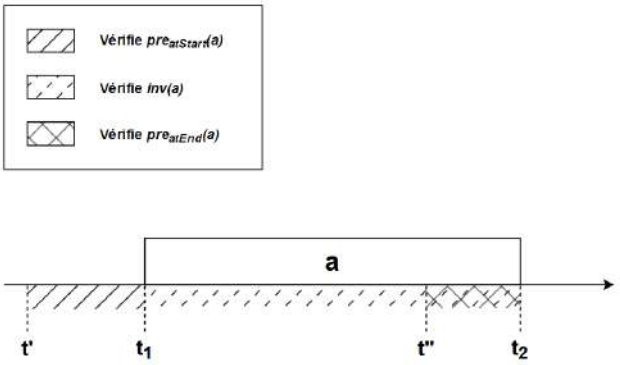

Figure 3.9. : Représentation des intervalles sur lesquels les trois types de conditions temporelles doivent être vérifiés.

— $pre_{atEnd}(a)$ *doit être vérifié sur un intervalle* $]t'', t_2[$ *avec* $t'' < t_2$

Une représentation des intervalles sur lesquels ces conditions doivent être vérifiées est affichée dans la Figure 3.9.

La définition temporelle des actions introduites dans PDDL 2.1 implique également des modifications de formalisme quand elles sont appliquées à la planification hiérarchique. En effet, en planification temporelle et hiérarchique, les tâches primitives et abstraites sont définies sur des intervalles plutôt que comme des évènements instantanés. Ainsi, les contraintes s'appliquant sur ces tâches ne sont simplement plus les contraintes ponctuelles (soit $<$, $\leq$ et $=$), mais l'ensemble des contraintes s'appliquant entre intervalles. Ces contraintes sont appelées les *contraintes de Allen* [All83]. Une visualisation de ces contraintes est affichée dans la Figure 3.10. En HDDL2.1, ces contraintes temporelles sont modélisées par le fait que les contraintes entre tâches s'appliquent soit sur les tâches entières, soit sur les points de début et de fin des tâches. Un exemple de méthode temporelle HDDL2.1 est représenté sur le Listing 3.9. Cet exemple représente une méthode modélisant un passage de relais entre deux coureurs. Pour effectuer le passage de relais, le premier coureur doit tenir le relais, puis le second doit également l'attraper, puis le premier coureur doit le lâcher. En terme de planification, cela se modélise par le fait que la tâche du premier coureur (tenir le relais) doit se terminer *après le début* de la tâche du second coureur.

Enfin, l'utilisation d'actions temporelles a également une incidence sur la définition d'un plan solution temporel. La différence tient au fait que les actions d'un plan temporel sont assignées à un instant de temps, qui détermine leur date de début. Ainsi on peut définir un plan temporel de la façon suivante :

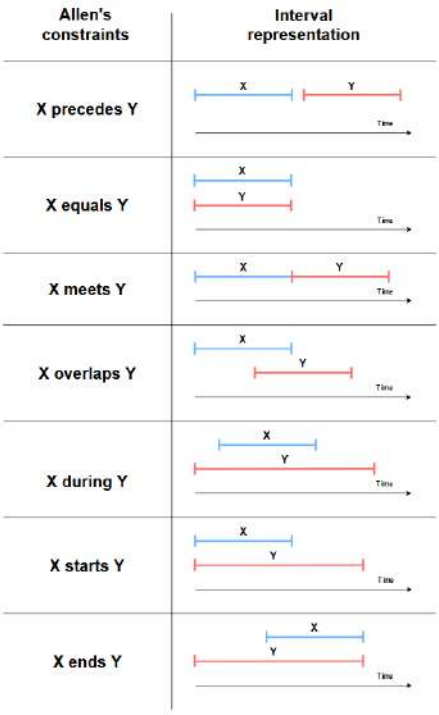

Figure 3.10. : Représentation des différentes contraintes possibles entre intervalles.

```
(:method passage_relais
    :parameters (?c1 ?c2 - Courreur)
    :task (changer_courreur ?to)
    :precondition ()
    :subtasks (and
        (t1 (tenir_relais ?c1))
        (t2 (tenir_relais ?c2))
      )
    :ordering (and
      (< (start t2) (end t1))
    )
)
```

Listing 3.9 : Déclaration d'une méthode temporelle HDDL2.1.

Définition 9 *Un **plan temporel** $\pi = \langle (a_1, t_1), (a_2, t_2), \ldots, (a_n, t_n) \rangle$ est un ensemble de couples actions/dates où $\forall i \in [\![1, n]\!]$, $a_i = (nom(a_i), start(a_i), end(a_i), inv(a_i), d_{a_i})$ est une action temporelle et t_i est la date de début de a_i.*

Un plan temporel est un plan solution d'un problème de planification classique (resp. hiérarchique) si le plan est exécutable dans l'état initial (toutes conditions des actions sont vérifiées tel que dans la Définition 8) et que l'état final du plan contient l'état objectif (resp. le plan est issu de la décomposition du réseau de tâches initial dans le cadre hiérarchique).

L'annexe A contient un exemple de domaine hiérarchique et temporel écrit en HDDL2.1 pour un problème de chasse aux mines. Une instance associée à ce domaine est également disponible dans cette annexe.

3.3 Classification des planificateurs existants

Dans ce paragraphe, nous proposons de faire une revue des approches de planification hiérarchique existantes dans la littérature. Ces approches sont d'une grande diversité. C'est pourquoi nous proposons de classifier ces approches en fonction de deux paramètres. Ces deux paramètres sont (1) la catégorie de problèmes pouvant être résolus et (2) le type de solution renvoyé par le planificateur.

La suite de ce paragraphe est organisée de la manière suivante : nous présentons dans un premier temps la classification utilisée pour présenter les planificateurs existants, puis nous présentons successivement les différentes catégories de planificateurs.

3.3.1 Classes de problèmes et de solutions

Comme évoqué précédemment, la classification que nous proposons repose sur (1) le type de problème résolu et (2) l'expressivité du plan solution renvoyé.

Trois classes de problèmes Pour ce qui est de la classification des problèmes résolus, nous utilisons les trois classes de problème mises en évidence par Cushing [Cus+07]. La caractérisation de ces classes de problème se fait en fonction du niveau de *concurrence* requis dans la solution. Dans le domaine de la planification, le mot "concurrence" est synonyme de "en même temps". En d'autres termes, on dit que deux actions sont *concurrentes* si leur exécution est réalisée en simultanée. À l'inverse, on dit que deux actions sont *séquentielles* si elles s'exécutent l'une après l'autre. De la même façon, un plan solution est dit *séquentiel* si toutes ses actions sont planifiées séquentiellement et *concurrent* si au moins deux de ses actions sont concurrentes.

Avant de poursuivre, il convient de définir la concurrence. En effet, jusqu'à présent, nous avons présenté un plan solution comme une séquence d'actions, temporelles ou non, réalisant un état objectif et/ou un ensemble de tâches hiérarchisées. Il est cependant possible d'étendre la notion de plan solution pour permettre à plusieurs

actions de s'exécuter simultanément. Cette notion de concurrence est particulièrement naturelle. Par exemple, dans le cas d'une flotte robotique marine, les appareils peuvent opérer dans plusieurs zones. Cependant, toutes les actions ne sont pas réalisables en même temps. Par exemple, un même appareil ne peut pas se déplacer à deux endroits en même temps. Ou encore, deux appareils ne peuvent pas effectuer la même tâche simultanément.

En pratique, un critère permettant de déterminer si deux *changements d'état* peuvent s'effectuer simultanément a été présenté par GraphPlan [BF97]. Un changement d'état représente une modification de l'état du problème. Dans le cadre de la planification non temporelle, un *changement d'état* est équivalent à une action. Cependant, dans le cadre de la planification temporelle, chaque action comprend deux changements d'état, un au début de l'action et un à la fin. Deux changements d'état peuvent être concurrents si l'effet du premier changement n'interfère pas avec les préconditions ou les effets du deuxième changement. On dit alors que les changements d'état sont mutuellement indépendants. Plus formellement, on a la définition suivante :

Définition 10 *Soient* $c_1 = (pre(c_1), add(c_1), del(c_1))$ *et* $c_2 = (pre(c_2), add(c_2), del(c_2))$ *deux changements d'état, on dit que* c_1 *et* c_2 *sont mutuellement indépendants si :*

$$\begin{cases} del(c_1) \cap (pre(c_2) \cup add(c_2)) = \emptyset \\ del(c_2) \cap (pre(c_1) \cup add(c_1)) = \emptyset \end{cases} . \tag{3.1}$$

Dans le cas contraire, on dit que c_1 *et* c_2 *s'excluent mutuellement.*

Ainsi, dans le cadre temporel ou non, deux actions peuvent s'exécuter en concurrence si (1) leurs conditions d'exécution sont vérifiées et si (2) les changements d'état simultanés sont tous mutuellement indépendants.

Il faut cependant noter que le cadre temporel a une notion de la concurrence plus complexe que le cadre non temporel. Effectivement, il y a de nombreuses façons pour que deux actions temporelles s'exécutent simultanément. Deux actions peuvent être parfaitement synchronisées (leurs débuts et fins sont simultanés), ou se recouvrir partiellement, ou encore l'une peut contenir l'autre. Plus simplement, si deux actions temporelles a_1 et a_2 s'exécutent respectivement sur les intervalles $[s_{a_1}, e_{a_1}]$ et $[s_{a_2}, e_{a_2}]$, alors a_1 et a_2 sont concurrentes si $[s_{a_1}, e_{a_1}] \cap [s_{a_2}, e_{a_2}] \neq \emptyset$. Les différentes formes de concurrence sont aussi nombreuses que les contraintes entre intervalles, les différentes configurations sont donc représentées par les contraintes de Allen (voir Figure 3.10).

La planification d'actions concurrentes est connue pour être plus complexe que la planification d'actions séquentielles. En effet, autoriser la concurrence augmente, de fait, le nombre d'actions pouvant être planifiées à un même instant, ce qui augmente d'autant la complexité du problème. Ainsi, le degré de concurrence nécessaire à la résolution du problème est un indicateur de la difficulté de résolution du problème. Dans cette optique, la classification de Cushing définit trois catégories de problème :

— **Catégorie 1 :** Comprend l'ensemble des problèmes dont toutes les solutions sont séquentielles. On parle de problèmes *intrinsèquement séquentiels*.
— **Catégorie 2 :** Comprend l'ensemble des problèmes dont les solutions peuvent être soit séquentielles, soit concurrentes. On parle de problèmes *mixtes*.
— **Catégorie 3 :** Comprend l'ensemble des problèmes dont toutes les solutions sont concurrentes. On parle de problèmes *intrinsèquement concurrents*.

Nous proposons de classifier les approches de planification suivant leur capacité à résoudre des problèmes de ces trois catégories. En effet, un planificateur capable de résoudre des problèmes dans la troisième catégorie sera également capable de résoudre les problèmes des deux précédentes catégories (qui nécessitent une expressivité moindre). À l'inverse, certains planificateurs se limitent à des problèmes de plus faible catégorie en échange d'un gain d'efficacité algorithmique.

En réalité, il a été démontré que la planification temporelle de problèmes de première catégorie est équivalente à de la planification non temporelle avec un coût d'exécution pour les actions [Cus+07]. Un planificateur temporel se limitant à la première catégorie de problème peut directement appliquer des techniques de la planification non temporelle. À l'opposé, il a été montré qu'un problème de troisième catégorie est nécessairement un problème temporel [Cus+07]. L'ajout de la dimension temporelle dans les problèmes de planification implique donc de nouvelles contraintes et requiert des méthodes de résolution plus fines de la part du planificateur.

Des exemples de problèmes de chaque catégorie sont représentés dans les Figures 3.11, 3.12 et 3.13. La Figure 3.11 représente un problème où un unique AUV doit réaliser l'image de deux zones. Comme il est le seul agent opérant, tout plan solution est nécessairement séquentiel (le problème est donc intrinsèquement séquentiel). Dans la Figure 3.12, l'objectif est le même que dans l'exemple précédent, mais un AUV disponible supplémentaire. Le plan solution précédent reste valide pour ce nouveau problème, mais l'ajout d'un nouvel AUV permet également des plans solution où les tâches sont réalisées en concurrence. C'est donc un problème de

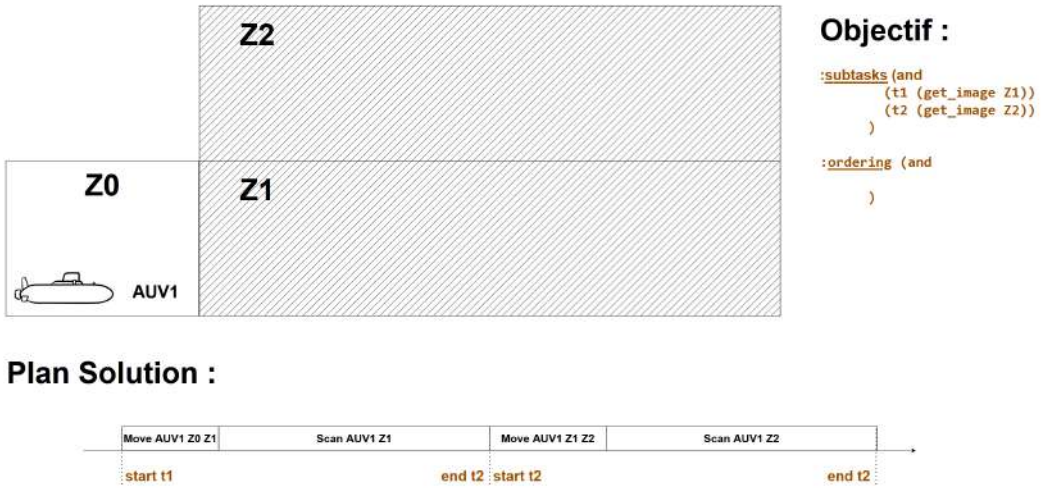

Figure 3.11. : Un exemple de problème dans la première catégorie de Cushing, toute solution est séquentielle.

deuxième catégorie. Enfin, la Figure 3.13 présente un problème où les contraintes sur le début et la fin des tâches induisent un plan nécessairement concurrent. Nous avons donc un exemple de problème de troisième catégorie.

Il faut noter que la concurrence du problème n'est pas nécessairement due à des contraintes hiérarchiques. Elle peut également être due à la satisfaction de conditions d'exécution pour des actions temporelles, et donc dans un contexte de planification classique. Un exemple d'actions temporelles de cette nature a été donné originellement par Cushing [Cus+07], il est représenté dans la Figure 3.14. Dans cet exemple, deux actions a et b doivent être planifiées. La condition invariante de a (celle qui doit être vérifiée tout au long de son exécution) est un effet du début de b. Démarrer b permet donc l'exécution de a. Symétriquement, la condition invariante de b est produite par le début de a. La seule solution pour pouvoir exécuter a et b est donc de démarrer (et de terminer) ces deux actions simultanément. Nous sommes donc bien en présence d'un problème de troisième catégorie dont la concurrence ne dépend pas de contraintes hiérarchiques.

Deux classes de plans solutions Le deuxième paramètre de notre classification des planificateurs est le *type* de plan solution retourné. En effet, jusqu'à présent nous avons défini un plan solution comme un ensemble d'actions dont les débuts sont associés à un certain instant temporel, on parle alors de *plan fixe*. Certains planificateurs sont capables de déterminer les *relations de causalité* entre les actions du plan solution et de produire un *ordonnancement relatif* des actions entre elles. Tout plan solution respectant cet ordonnancement offre alors la garantie d'être un

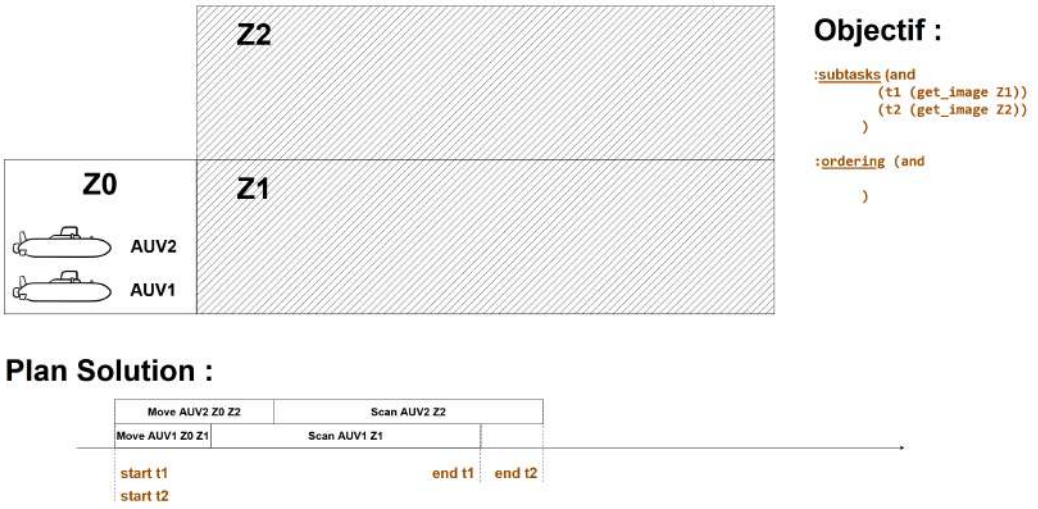

Figure 3.12. : Un exemple de problème dans la deuxième catégorie de Cushing, il admet des solutions séquentielles ou concurrentes.

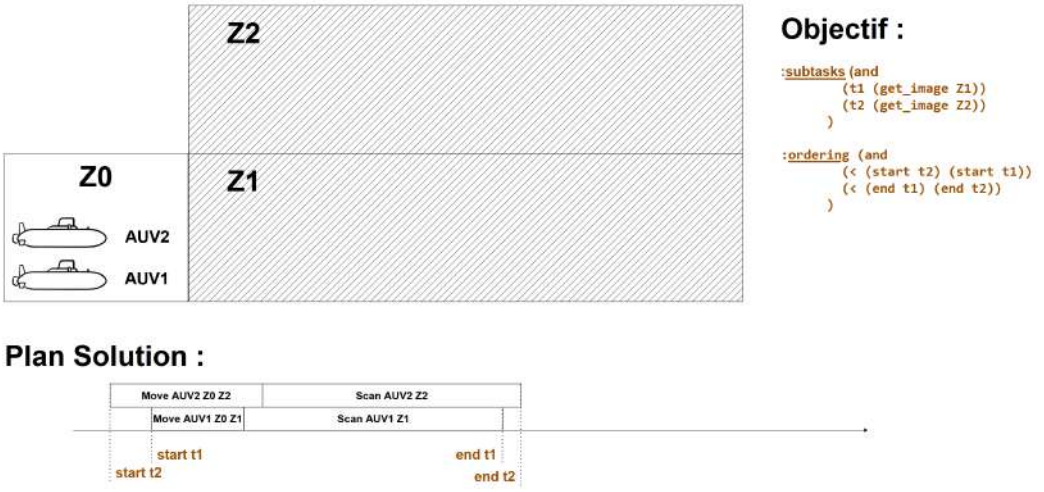

Figure 3.13. : Un exemple de problème dans la troisième catégorie de Cushing, toute solution est concurrente.

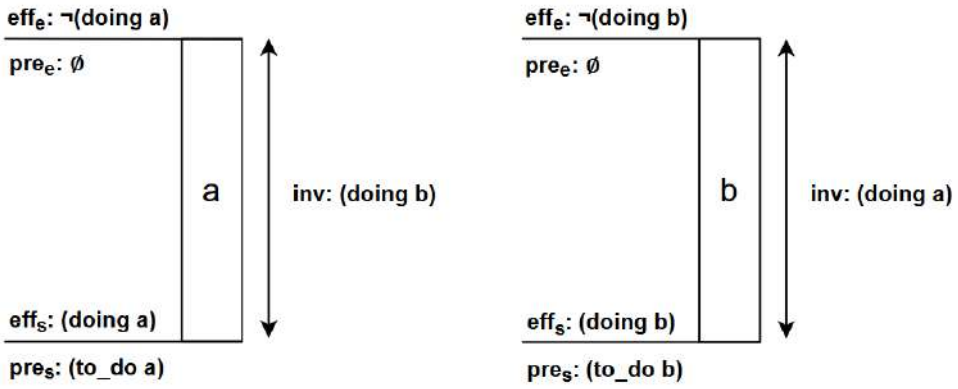

Figure 3.14. : Exemple de deux actions nécessairement concurrentes.

plan valide solution du problème. On parle alors de *plan partiellement ordonné*. Un exemple de plan partiellement ordonné avec ses plans fixes associés est représenté dans la Figure 3.15.

Les plans partiellement ordonnés présentent plusieurs avantages par rapport aux plans fixes. Tout d'abord, les plans partiellement ordonnés représentent la *structure causale* d'un plan solution. Ce qui signifie qu'en cas d'échec, lors de l'exécution d'une action, il est aisé de déterminer la partie du plan qui sera concernée par cet échec. Cet aspect des plans partiellement ordonnés permet des mécanismes de réparation et de fusion de plans plus efficaces.

Un plan partiellement ordonné est donc plus expressif qu'un plan fixe. En ce sens, le type de plan solution produit est un indicateur de l'expressivité d'un planificateur. Nous proposons donc de classifier les approches de planification suivant les catégories suivantes :

— **Catégorie f :** Pour les approches produisant des *plans solutions fixes*.
— **Catégorie p :** Pour les approches capables de produire des *plans solution partiellement ordonnés*.

Il faut cependant noter que les techniques de recherche de solutions sont différentes dans le cas où le plan recherché est partiellement ordonné ou fixe. Un plan partiellement ordonné étant plus informatif qu'un plan fixe, il y a généralement un coût algorithmique supérieur à la création d'un tel plan. Par ailleurs, les algorithmes produisant de tels plans sont souvent moins efficaces.

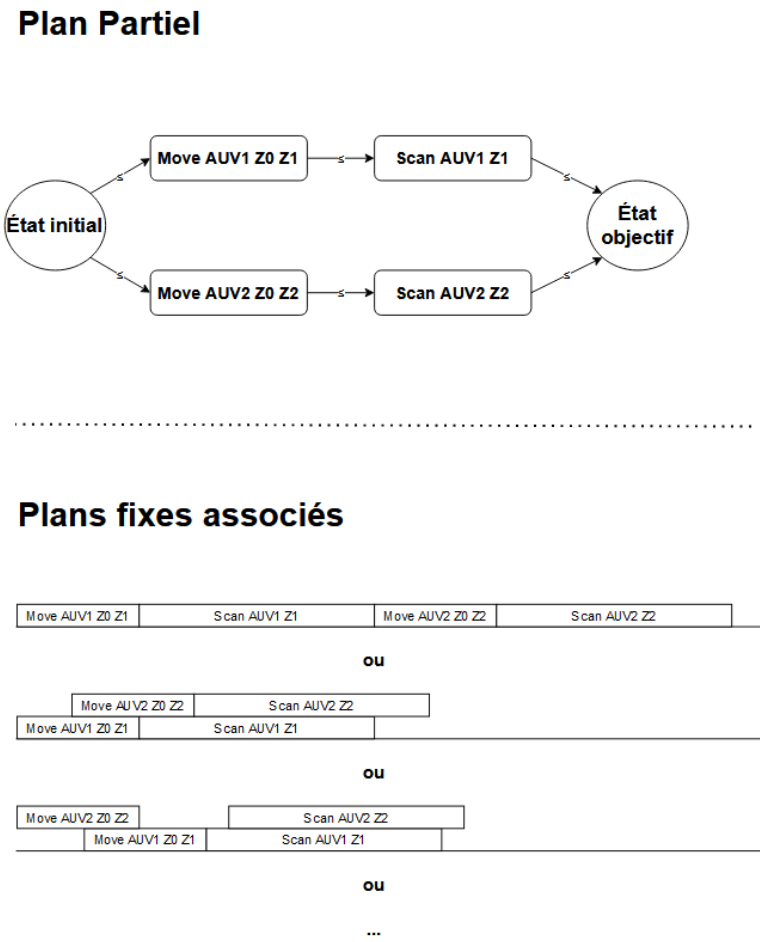

Figure 3.15. : Exemple de plan partiellement ordonné ainsi que quelques plans fixes qui lui sont associés.

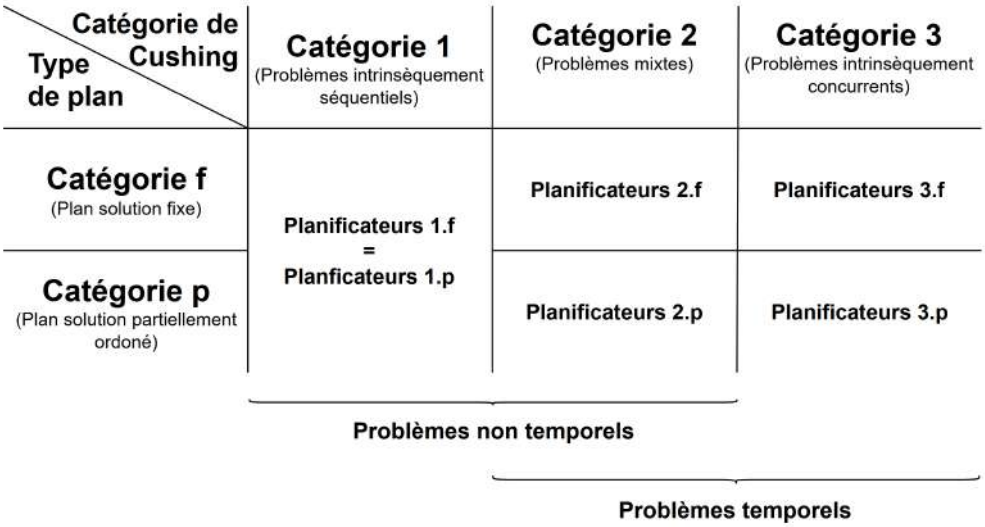

Figure 3.16. : Classification des approches de planification en fonction du type de problème résolu et du type de solution produit.

Résumé de la classification Nous avons présenté les deux paramètres de notre classification des approches de planification hiérarchique. Par exemple, les planificateurs de la catégorie *2.f* sont des planificateurs capables de résoudre des problèmes jusque dans la deuxième catégorie de Cushing en produisant un plan solution fixe. L'exception étant que les catégories *1.f* et *1.p* sont confondues. En effet, dans le cadre d'une solution séquentielle, un plan partiellement ordonné admet une unique linéarisation. Un plan partiellement ordonné n'a donc pas d'expressivité additionnelle comparée à un plan fixe.

De plus, du fait des différences entre les formalismes de planification classique et temporelle, nous distinguerons les planificateurs résolvant des problèmes classiques ou temporels. Il faut noter que les problèmes temporels de première catégorie de Cushing sont équivalents à des problèmes classiques de première catégorie également. Les problèmes de première catégorie sont donc tous identifiés comme des problèmes non temporels[Cus+07]. À l'opposé, il a été montré que les problèmes de troisième catégorie sont nécessairement des problèmes temporels [Cus+07].

Un tableau récapitulatif des différentes catégories est représenté dans la Figure 3.16.

3.3.2 Catégorie 1 : Problèmes intrinsèquement séquentiels

Dans ce paragraphe, nous nous intéressons aux approches de planification hiérarchique ayant été proposées dans le but de résoudre des problèmes de la première catégorie de Cushing. Ces problèmes ont pour objectif d'accomplir un ensemble *totalement ordonné* de tâches. Dans ce cadre, l'objectif de la planification est donc de trouver comment décomposer la séquence de tâches initiales afin d'obtenir une séquence d'actions applicable à l'état initial. Dans ce type de problème, *l'ordonnancement* des actions est induit par la décomposition des tâches abstraites, et n'a pas à être déterminé par le planificateur. L'objectif est donc uniquement de trouver une décomposition résultant en une séquence valide. Les approches produisant des plans partiellement ordonnés visent quant à elles à trouver un ordonnancement des actions garantissant leur exécutabilité. Elles ne sont donc pas adaptées à ce type de problème.

À la place, deux familles d'approches ont été proposées afin de résoudre ces problèmes. La première famille essaie de résoudre récursivement la dernière tâche du réseau (totalement ordonné). Si la tâche est *primitive*, et que ces préconditions sont vérifiées dans l'état courant, alors l'action est retirée du réseau de tâches et ses effets appliqués dans l'état courant. Si la tâche est *abstraite*, elle est résolue en appliquant une méthode dont les préconditions sont également vérifiées dans l'état courant. Pour ces approches, le planificateur doit uniquement choisir les *méthodes* à appliquer pour la décomposition des tâches. À notre connaissance, la première version de ce type de planificateur est HATP [Mon+07]. L'objectif de ce planificateur est d'utiliser la structure HTN du problème pour (1) pouvoir introduire rapidement des séquences d'actions dans le plan et (2) être capable d'expliquer la tâche abstraite étant réalisée. Le planificateur a reçu des améliorations successives [DLA15 ; LSA18] qui ont permis des gains d'efficacité du planificateur. Le planificateur GDP-h [Shi+12] a la particularité d'adapter une heuristique issue de la planification classique (non hiérarchique) afin de sélectionner la décomposition des tâches efficacement. Enfin, le planificateur GTPyhop [Nau+21] offre un cadre de planification Python permettant de réaliser le test d'heuristiques pour les problèmes totalement ordonnés. Sa spécificité est de pouvoir résoudre des problèmes de planification classique en exprimant un état objectif comme une séquence de tâches abstraites.

La deuxième famille d'approches résolvant des problèmes hiérarchiques totalement ordonnés considère ces problèmes comme des problèmes de satisfiabilité booléenne [Coo23]. Le principe général de ces planificateurs est d'encoder le réseau de tâches ainsi que ses décompositions possibles sous la forme d'une formule SAT dont les

variables de décision portent sur le choix des méthodes de décomposition. L'encodage SAT du problème nécessite de limiter la taille du plan solution à une borne supérieure qui est itérativement augmentée si aucune solution n'est trouvée. Ces approches reposent sur des solveurs SAT et leur efficacité dépend donc uniquement de la modélisation SAT du problème. Une première version de ces approches a été proposée par totSAT [BHB18], qui a ensuite été améliorée successivement par Tree-REX [Sch+19] et ensuite par Lilotane [Sch21a]. Ce dernier planificateur, Lilotane, est à l'heure actuelle le planificateur le plus efficace pour ce type de problème [BHB21].

3.3.3 Catégorie 2 : Problèmes mixtes

À présent, nous allons discuter des approches de planification permettant de résoudre les problèmes de la deuxième catégorie de Cushing. Comme représenté par la Figure 3.16, ces approches se divisent en deux sous-catégories. La première cherche à produire des plans fixes quand la seconde cherche à générer des plans partiellement ordonnés. Les algorithmes utilisés par les approches des deux catégories sont de nature très différente et pour cette raison, nous séparons ces deux types d'approches.

Catégorie 2.f : Approches en plans fixes Les approches fixes de la deuxième catégorie sont toutes basées autour d'une idée centrale. Les tâches *sans prédécesseurs* du réseau de tâche sont résolues par rapport à un état courant. Pour une tâche primitive cela signifie (1) vérifier que les préconditions de l'action sont vérifiées et (2) appliquer ses effets à l'état courant. Pour une tâche abstraite, cela signifie la décomposer selon une méthode dont les préconditions sont vérifiées dans l'état courant. On parle d'approches *basées sur l'espace d'états*. Un schéma représentant ce procédé est représenté dans la Figure 3.17. À la différence des approches de la catégorie 1.f et 1.p, basées sur le même principe (voir le paragraphe 3.3.2), plusieurs tâches sans prédécesseurs peuvent exister. Les approches de cette catégorie doivent donc choisir (1) quelles méthodes utiliser pour la décomposition (à l'instar des méthodes de la première catégorie) et (2) dans quel *ordre* résoudre ces tâches.

Le premier planificateur à avoir présenté et démontré cette idée est le planificateur UCMP [EHN94]. Le principal intérêt de cette approche est d'avoir fourni la formalisation ainsi que la démonstration de la complétude et de la correction des approches basées sur l'espace d'état. Quelques années plus tard, une nouvelle approche nommée SHOP2 [Nau+03] simplifie et optimise le formalisme de UCMP. Ce

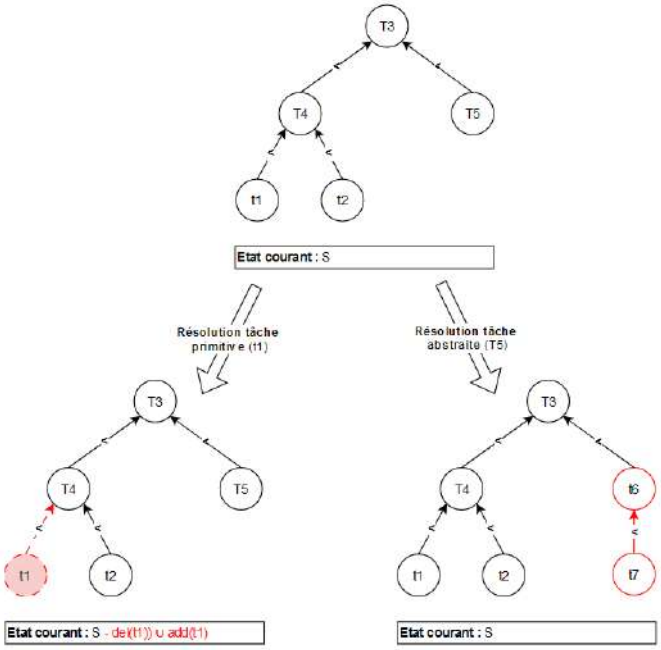

Figure 3.17. : Principe de résolution dans l'état courant de tâches primitives et abstraites.

planificateur va rapidement s'imposer comme un nouveau standard pour la planification hiérarchique. Les planificateurs de sa catégorie étendront alors son formalisme et perfectionneront ses techniques de recherche de solution.

Par exemple, C-SHOP [BK04] étend l'algorithme de SHOP2 afin de prendre en compte les milieux partiellement observables et/ou incertains. Le planificateur ND-SHOP [KN04] ainsi que sa seconde version YoYo [Kut+05] permettent de planifier dans des milieux où les effets des actions sont non déterministes. Le planificateur HTNPlan-P [SBM09] permet d'ajouter des préférences dans la manière d'accomplir les tâches. Enfin, Hy-CIRCA [Gol+16] mène un processus de planification en deux temps, où le plan originellement produit via SHOP est augmenté avec des variables et des contraintes supplémentaires.

Pour ce qui est des techniques de recherche de solutions, de nombreuses approches ont également été publiées. Une idée répandue est d'utiliser une recherche arborescente de Monte-Carlo [Cha+08] pour la sélection de la tâche à résoudre. Cette idée a d'abord été explorée par UCT [KS06], où un algorithme d'apprentissage est couplé à la recherche Monte-Carlo. Ce type de sélection est aussi utilisé dans les planificateurs Duet [Ger+08], AHTN [OB15] et AHTNCO [Lin+20], ou encore RAEPlan [Pat+19]. La particularité de AHTN, et de son amélioration AHTNCO, est d'effectuer une recherche antagoniste (adversarial search), où deux planificateurs cherchent en concurrence un plan solution. RAEPlan quant à lui arrête l'exploration

Monte-Carlo à une profondeur donnée par un paramètre défini au préalable, ce qui lui permet de gagner en efficacité.

D'autres approches ont fait le choix d'avoir de l'apprentissage. Ces approches ont pour but de s'améliorer au fil des exemples. Par exemple, Q-SHOP [HKM10] utilise un algorithme d'apprentissage par renforcement pour sélectionner les tâches à résoudre et les méthodes de décomposition. Une heuristique apprenante a également été implémentée pour le planificateur RAEPlan, nommée UPOM [Pat+20]. Ou encore, HyperTensioN [MMD20] utilise des techniques de *dejavu* pour reconnaître des situations déjà rencontrées dans de précédents problèmes.

D'autres approches basées sur SHOP2 peuvent également être notées. Dans Angelic [MRW08], un algorithme A* [RN10] est couplé à SHOP2 pour obtenir des plans optimaux. Inversement, SHPE [MJC14] cherche une première solution qui est ensuite optimisée progressivement. iSHOP [Ram+16] utilise des techniques d'instanciation pour simplifier le problème à résoudre.

Le planificateur LAMA [RW10] utilise des *landmarks* afin de guider la recherche d'une solution. Ces landmarks peuvent être vus comme un ensemble ordonné de sous-objectifs devant être remplis afin de résoudre le problème global. Ils sont utiles soit pour éliminer les méthodes ne permettant pas la réalisation des landmarks soit pour diviser le problème en sous-problèmes. La génération automatique des landmarks est basée sur la méthode proposée dans [HPS04]. Ce principe de résolution a ensuite été réutilisé par les planificateurs GoDeL [Shi+13] et HOpGDP [Shi+16].

L'algorithme SHOP2 [Nau+03] a également été adapté pour une version limitée de la planification temporelle. Les actions temporelles sont ajoutées en séquence dans un plan solution. Ainsi si SHOP2 et ses dérivés sont capables de résoudre des problèmes temporels, ils ne sont uniquement capables d'apporter une solution séquentielle à ces problèmes.

Pour résumer, ces approches sont basées sur un même algorithme de planification SHOP2 [Nau+03] qui a ensuite été progressivement amélioré et étendu pour inclure des contraintes et des situations supplémentaires. Notons enfin qu'une version restructurée et en source ouverte de SHOP2, SHOP3 [GK19], a été présentée, facilitant son utilisation et les développements.

Une autre façon de résoudre un problème de planification mixte est de l'*encoder* en un problème STRIPS. Ce type d'approche a le bénéfice de pouvoir utiliser les planificateurs et heuristiques développés dans le cadre de la planification STRIPS pour la résolution de problème HTN. Cependant, la planification hiérarchique étant strictement plus expressive que la planification STRIPS [EHN96], ce type d'encodage

souffre nécessairement de restrictions, soit sur le type de problèmes résolus, soit sur la solution des problèmes résolus. Le premier exemple de ce type d'encodage a été proposé dans le cadre des problèmes totalement ordonné [AKN09]. Bien que limitée, cette approche s'est révélée prometteuse et fut développée par HTN2STRIPS [Alf+16] pour résoudre les problèmes partiellement ordonnés. Ces encodages restent malgré tout inférieurs aux approches précédentes (basées sur SHOP), jusqu'à ce qu'une nouvelle version, HTN2SAS [Beh+22] rende ce type d'approche compétitive avec les autres planificateurs de sa catégorie. À l'instar des encodages SAT (voir le paragraphe3.3.2), les encodages HTN2STRIPS et HTN2SAS réalisent une recherche incrémentale. Ils limitent d'abord la taille maximale de la solution, puis l'augmentent en cas d'échec. Il faut finalement noter qu'à l'heure actuelle, ces approches ne sont pas capables de résoudre des problèmes temporels.

Catégorie 2.p : Approches en plans partiellement ordonnés Les approches produisant des plans partiellement ordonnés sont basées sur une tout autre idée que les approches fixes. En effet, au lieu de résoudre successivement des tâches sur un état courant, elles cherchent à raffiner et résoudre les *défauts* (ou flaws) d'un plan partiel initial. Ces raffinements peuvent être de trois types :

— **Décomposition :** L'une des tâches abstraites du réseau de tâches est décomposée selon une méthode du problème.
— **Ajout de lien causal :** Un lien causal est une relation de causalité qui est ajoutée au plan. Ces relations relient deux actions ainsi qu'une proposition. La première action produit la proposition par son effet et la seconde la nécessite comme précondition. Le but d'un lien causal est de *protéger* la proposition entre les deux actions.
— **Menace causale :** On dit qu'un lien causal peut être *menacé* par une troisième action m, consommant la propriété protégée par le lien causal. Une relation d'ordre est introduite et contraint m à être exécutée soit avant le lien causal, soit après.

Ainsi, le plan partiel initial est successivement raffiné jusqu'à ce que plus aucun défaut ne soit présent dans le plan partiel. Le plan partiel est alors un plan solution du problème. On parle alors de planification POCL (pour *Partial Order Causal Link*). Une représentation de ces trois types de raffinement est représentée dans la Figure 3.18.

Les approches produisant des plans partiellement ordonnés ont connu plusieurs phases de développement. Les premiers systèmes ayant incorporé ces techniques de

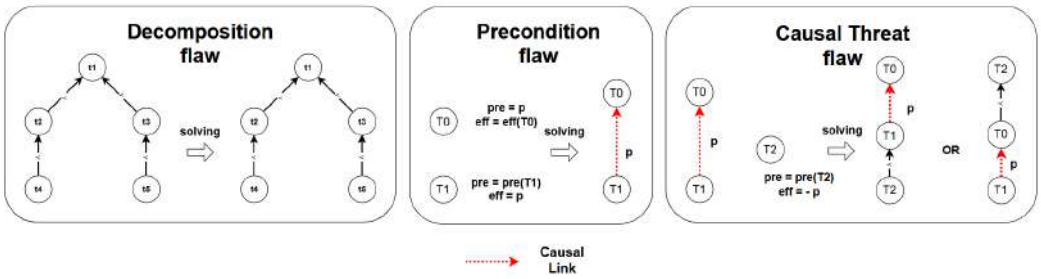

Figure 3.18. : Les trois types de raffinement possibles dans un plan partiel.

planification sont NOAH [Sac75] et NONLIN [Tat77]. Ces planificateurs ont permis de définir un premier essai de formalisation et de résolution de problèmes avec des plans partiellement ordonnés.

Quelques années plus tard, une seconde vague de ces planificateurs va permettre d'apporter les preuves de complétude et de correction de cet algorithme. Dans cette seconde vague, on peut citer O-Plan [CT91], mais aussi SIPE [Wil84]. Ces systèmes ont également été étendus dans le but de pouvoir résoudre des problèmes temporels : SIPE2 [Wil90] et O-Plan2 [TDK94]. Bien que ces planificateurs soient capables de gérer l'intégralité des contraintes temporelles (les contraintes de Allen), ils ne sont pas basés sur des formalismes suffisamment expressifs pour exprimer l'intégralité des problèmes temporels.

Suite à ces planificateurs, la communauté va progressivement perdre de son intérêt pour les plans partiellement ordonnés, qui sont alors considérés comme trop coûteux à produire, et moins efficaces que les approches fixes.

Ce n'est que récemment que l'intérêt va être ravivé. En effet, les systèmes HiPOP [Bec+14] et PANDA [Sch15] vont présenter de nouvelles heuristiques permettant de rétrécir l'écart entre les méthodes fixes et les plans partiellement ordonnés. L'architecture de contrôle HiDDeN [GLB12] va quant à elle permettre de démontrer les qualités de replanification des plans partiellement ordonnés. Depuis, de nombreux planificateurs produisant de tels plans sont présentés. Par exemple, HEART [GMA18] génère le plan solution au fur et à mesure de l'exécution de la mission. Ou encore, GDPOP [WC18] permet une résolution optimale du problème. Enfin, les systèmes HiPOP et PANDA ont connu des mises à jour fréquentes dont les nouveautés ont été présentées respectivement dans [LA21] et [Höl+21].

L'intérêt pour les systèmes en plans partiellement ordonnés vient également du fait que ces systèmes sont capables de résoudre l'intégralité des contraintes temporelles. Cette particularité fait qu'ils sont majoritairement utilisés pour résoudre des pro-

blèmes temporels de troisième catégorie, ce que nous verrons dans le paragraphe suivant.

3.3.4 Catégories 3 : Problèmes intrinsèquement concurrents

Nous allons à présent passer en revue les approches de planification hiérarchique capables de résoudre des problèmes de la troisième catégorie de Cushing. Ces problèmes n'admettent que des solutions donc au moins deux actions sont concurrentes. On parle alors de problèmes *intrinsèquement concurrents*. On relève deux types de concurrence nécessaire, soit la concurrence est due à des contraintes temporelles, soit elle est requise pour satisfaire les conditions d'exécution (temporelles) des actions du plan solution. Ce type de problème ne peut donc exister que dans le cadre de la planification temporelle.

Cette concurrence entraîne des complications pour les systèmes produisant des plans fixes qui ont été revus plus haut. En effet, lorsque deux actions nécessitent d'être exécutées en même temps, la planification de ces actions doit également se faire de manière commune. Or la plupart de ces approches ne permettent de résoudre qu'une seule tâche à la fois, et ne permettent donc pas d'introduire d'actions interdépendantes dans le plan solution. Ainsi, les systèmes cherchant à résoudre des problèmes de troisième catégorie nécessitent des techniques de résolution spécifiques.

Catégorie 3.f : Approches en plans fixes Les exceptions à cela sont les planificateurs SIADEX [Asu+05 ; JJP15] et VHPOP [YS03]. Ces approches sont fixes et basées sur l'algorithme de résolution SHOP2. Afin de traiter la concurrence nécessaire, les actions interdépendantes temporelles sont précompilées en *macro-actions* (une macro-action est la composition de plusieurs actions) qui ne sont pas inderdépendantes d'autres actions. On parle alors de *single hard envelopes* [Col+09] (ou SHE). Un exemple d'une telle compilation est représenté dans la Figure 3.19. Ces approches se limitent aux problèmes de troisième catégorie dont une telle reformulation en SHE est possible. Elles ne peuvent donc pas résoudre tous les problèmes de la troisième catégorie, mais sont particulièrement efficaces sur les problèmes qu'elles peuvent reformuler [JJP15].

Catégorie 3.p : Approches en plans partiellement ordonnés Pour résoudre les problèmes de concurrence nécessaire, l'approche préférée a d'abord été émise par un

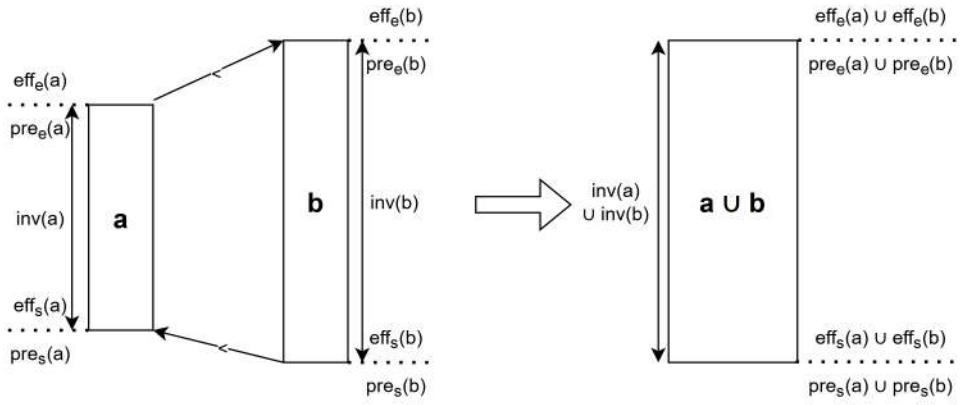

Figure 3.19. : Compilation de deux actions temporelles nécessairement concurrentes en une macro action représentant l'exécution des deux actions.

planificateur non hiérarchique dénommé IxTeT [Lem04]. Ce planificateur introduit la notion de *Timeline*, qui permet de représenter la valeur des propositions du problème en fonction du temps. Un plan solution est alors représenté par un ensemble de *Timelines* (une pour chaque proposition). L'objectif devient alors, à l'instar des approches POCL (voir le paragraphe 3.3.3), de résoudre les défauts des différentes Timelines jusqu'à ce que toutes les contraintes du plan partiel soient vérifiées. Le plan partiel est alors un plan solution du problème.

Ce type d'approche est la plus répandue parmi les planificateurs de la troisième catégorie. Une première version a été proposée par EUROPA [Bar+12], où les Timelines sont utilisées à la fois pour modéliser les propriétés booléennes et les propriétés numériques du problème. Cependant, ce type d'approche en Timelines a été perfectionné par les systèmes FAPE [Dvo+14] et CHIMP [Sto+15]. Ces deux planificateurs sont similaires dans leur représentation du problème, mais diffèrent quant à l'heuristique employée pour l'exploration de l'espace de recherche. En effet, quand FAPE utilise une heuristique issue de la planification POCL (voir le paragraphe 3.3.3), CHIMP utilise une heuristique meta-CSP afin de résoudre les conflits du plan partiel. Il faut enfin noter que le planificateur FAPE a reçu de récentes mises à jour permettant encore d'améliorer ses capacités, notamment dans son efficacité à représenter les problèmes [Bit+20].

3.3.5 Conclusions de l'état de l'art

Dans ce chapitre, nous avons passé en revue les différentes approches existantes dans le domaine de la planification hiérarchique. Ces approches peuvent être classifiées

suivant le type de problèmes qu'elles cherchent à résoudre. La première catégorie ne résout que les problèmes intrinsèquement séquentiels et est donc particulièrement adaptée à la planification pour un unique agent. La deuxième catégorie vise à résoudre les problèmes partiellement ordonnés, mais où la concurrence n'est pas requise. La plupart des approches de deuxième catégorie produisent cependant des plans séquentiels et fixes. Enfin, les approches de troisième catégorie sont soit (1) basées sur une recompilation du problème pour en retirer la concurrence nécessaire, soit (2) basées sur un formalisme *Timeline* proche du formalisme POCL. La solution (1) est efficace, mais se limite à une sous-catégorie des problèmes de troisième catégorie. Ainsi, les seules approches existantes capables de résoudre l'ensemble des problèmes de planification hiérarchique sont les approches Timeline.

Cependant, la majorité des recherches de la communauté ont principalement porté sur les problèmes de première et deuxième catégorie. Le problème de la chasse aux mines présente à la fois une forte interdépendance entre les agents, ainsi que des enjeux de concurrence très importants. L'objectif des travaux présentés dans cette thèse a donc été de développer des méthodes pour réutiliser les techniques de planification de première ou deuxième catégorie afin de répondre à des problèmes de troisième catégorie.

Deuxième partie

Contributions

Introduction

Au cours de cette thèse, divers travaux ont été réalisés. Ils ont porté sur différents aspects de la planification hiérarchique dans une démarche incrémentale. Les travaux ont commencé par porter sur la planification non temporelle, et ont ensuite incorporé les éléments de la planification temporelle. L'objectif de cette démarche est de comprendre les différences et spécificités de ces deux aspects de la planification et de déterminer dans quelles mesures les avancées dans l'un des deux domaines peuvent bénéficier à l'autre.

La première partie des travaux a porté sur un nouvel encodage, appelé **CTHD** (Concurrent Task Holder Decomposition), permettant d'exprimer un problème HTN (voir Définition 7) en un problème STRIPS (voir Définition 6), et ainsi de bénéficier de l'efficacité des planificateurs STRIPS existants pour la résolution de problème hiérarchique. La particularité de cet encodage est de construire une solution concurrente au problème HTN. Ces travaux ont donné lieu à une publication dans une conférence internationale [CPF23a].

La deuxième partie porte sur une nouvelle approche de planification **TEP** (*Hierarchical Temporal Events Planner*) de catégorie **3.p** (voir Figure 3.16) permettant de résoudre des problèmes temporels. La particularité de ce planificateur est d'utiliser les principes de résolution et les heuristiques des planificateurs non temporels, ce qui lui confère un avantage en termes d'efficacité. Ces travaux ont également fait l'objet d'une publication [CPF23b].

Enfin, la dernière partie des travaux a porté sur un encodage des problèmes hiérarchiques et temporels en problèmes hiérarchiques et non temporels. Cet encodage, nommé **TTC** (*Temporal Task Converter*) permet d'adapter le principe de résolution de TEP à n'importe quel planificateur hiérarchique non temporel, au prix d'une petite perte d'expressivité.

La Figure 3.20 représente les contributions au regard de la classification proposée au chapitre précédent. La suite de ce chapitre est organisée de la manière suivante, nous détaillons chacune de ces contributions successivement en présentant leurs aspects théoriques ainsi que les comparaisons qui ont été réalisées avec les approches existantes similaires.

Catégorie de Cushing / Type de plan	Catégorie 1 (Problèmes intrinsèquement séquentiels)	Catégorie 2 (Problèmes mixtes)	Catégorie 3 (Problèmes intrinsèquement concurrents)
Catégorie f (Plan solution fixe)		CTHD	
Catégorie p (Plan solution partiellement ordonné)		TTC	TEP

Figure 3.20. : Classification des contributions réalisées.

4 CTHD : Un encodage HTN vers STRIPS concurrent

Comme cela a été évoqué précédemment, les premiers travaux ont porté sur un encodage HTN vers STRIPS. L'objectif de cet encodage est d'écrire un problème STRIPS à partir d'un problème hiérarchique, tel que la solution du problème STRIPS décrit une solution du problème hiérarchique. Le problème hiérarchique peut ainsi être résolu en utilisant un planificateur STRIPS.

Ces travaux s'inscrivent dans la lignée de travaux précédents ayant montré des résultats prometteurs. Tout d'abord le principe général a été énoncé par le planificateur **TSTN** [AKN09]. Ce principe est de considérer le réseau de tâches du problème hiérarchique comme un objet de planification STRIPS. Cet objet STRIPS peut être modifié au moyen d'actions dont les effets imitent l'application d'un opérateur hiérarchique sur le réseau de tâches. Pour TSTN, qui se limite aux problèmes totalement ordonnés, il y a donc deux types d'opérateurs dans le problème STRIPS encodé : (1) des opérateurs de résolution de tâches primitives et (2) des opérateurs de résolution de tâches abstraites. Le planificateur STRIPS choisit quel opérateur appliquer au réseau afin de le résoudre entièrement. Il faut remarquer que la solution du problème STRIPS encodé est donc la séquence d'opérateurs à appliquer au réseau initial afin d'en obtenir une solution.

Représenter un réseau de tâches présente une limitation qui se retrouvera dans tous les encodages HTN vers STRIPS : le réseau a une taille maximale, appelée *progression bound*, définie au moment de l'encodage du problème. Les planificateurs utilisant ces encodages utilisent un processus itératif, encodant le problème à une *progression bound* donnée, puis en incrémentant cette progression bound si aucune solution n'est trouvée. De plus, la *progression bound* a une grande influence sur deux points :

— **L'existence d'une solution :** En effet, si la taille du réseau n'est pas suffisante, pour définir une décomposition valide, le problème encodé n'a pas de solution.
— **La complexité du problème à résoudre :** Si la taille du réseau est plus grande que nécessaire, un grand nombre d'opérateurs STRIPS inutiles seront introduits dans le problème encodé.

Le deuxième encodage HTN vers STRIPS ayant été proposé, nommé HTN2STRIPS [Alf+16], a permis de faire deux avancées majeures. Tout d'abord, ces travaux offrent une méthode permettant de déterminer un encadrement de la *progression bound*. Cet encadrement se limite aux problèmes *non récursifs* (aucune tâche abstraite ne peut se décomposer en elle-même) mais permet à la fois d'éviter les tailles de réseau trop petites pour permettre une solution, mais également de donner une condition d'arrêt au planificateur une fois la borne supérieure dépassée (et ainsi prouver la non-existence d'une solution). La seconde avancée apportée par HTN2STRIPS a été l'encodage (et la résolution) de problèmes hiérarchiques *partiellement* ordonnés. Pour cela, le réseau de tâches est modélisé dans le problème STRIPS en utilisant des propositions supplémentaires, caractérisant l'ordre dans lequel les tâches doivent être exécutées. Malgré tout, même si les problèmes résolus sont partiellement ordonnés, les solutions produites restent séquentielles.

Enfin la dernière version des encodages HTN vers STRIPS a été présentée sous le nom de HTN2SAS [Beh+22]. Cet encodage reprend le principe de résolution de HTN2STRIPS en introduisant une nouvelle façon de modéliser le réseau de tâches et ses raffinements afin de générer moins de propositions et d'actions dans le problème encodé STRIPS. Toutes ces améliorations ont permis de rendre les planificateurs par encodage compétitifs avec les planificateurs hiérarchiques les plus performants [Beh+22].

L'encodage présenté ici, CTHD, reprend les avancées introduites par TSTN, HTN2STRIPS et HTN2SAS tout en permettant de produire un plan solution *concurrent* pour le problème hiérarchique initial. Par les expérimentations, nous montrons que CTHD produit des plans solutions de meilleure qualité que les précédents encodages, le tout avec des temps de calcul comparables.

Dans la suite de cette section nous commençons par donner une formalisation du problème que nous cherchons à résoudre, puis nous présentons notre encodage, CTHD, enfin nous présentons les résultats des expérimentations que nous avons menées.

4.1 Définition du problème

Pour les problèmes STRIPS et hiérarchiques, nous utilisons respectivement les Définitions 6 et 7 (voir Section 3.2.1).

Soit P_h un problème de planification hiérarchique. $P_h = (L, \mathcal{T}, A, M, I, tn)$ est composé de L un ensemble fini de propositions logiques, de A un ensemble fini d'actions, de M un ensemble fini de méthodes, de $\mathcal{T}$ un ensemble fini de tâches (primitives ou abstraites), d'un état initial $I \subset L$, ainsi que d'un réseau initial de tâches tn.

Un réseau de tâches $tn = (T, \prec, \alpha)$ est lui-même composé d'un ensemble fini T de symboles représentant les tâches à résoudre, d'une relation d'ordre (partielle) $\prec$ sur les éléments de T, et d'une fonction $\alpha : T \mapsto \mathcal{T}$ associant chaque symbole de T à une tâche de $\mathcal{T}$. On dira qu'une tâche $t \in T$ *précède* une seconde tâche $t' \in T$ si $(t, t') \in \prec$. De même, on dira que $t \in T$ n'a pas de prédécesseur si $\forall t' \in T, (t', t) \notin \prec$.

Une tâche sans prédécesseur peut être résolue soit par l'application d'une action dans le cas d'une tâche primitive, soit par l'application d'une méthode dans le cas d'une tâche abstraite :

— Une action $a = (name(a), pre(a), add(a), del(a)) \in A$ résout une tâche $t \in T$ si (1) $name(a) = \alpha(t)$, si (2) $pre(a) \subset I$ et si (3) t n'a pas de prédécesseurs. Après la résolution de t, on obtient le problème suivant $P'_h = (L, \mathcal{T}, A, M, \gamma(I, a), tn' = (T', \prec', \alpha')$ avec :

  ◇ $\gamma(I, a) = (I - del(a)) \cup add(a)$

  ◇ $T' = T \setminus \{t\}$

  ◇ $\prec' = \{(t', t'') \in \prec \mid t' \neq t \text{ et } t'' \neq t\}$

  ◇ $\alpha' = \alpha \restriction_{T'}$ la restriction de α sur T'

— Une méthode $m = (task(m), pre(m), tn(m)) \in M$ résout une tâche $t \in T$ si (1) $task(m) = \alpha(t)$, si (2) $pre(m) \subset I$ et si (3) t n'a pas de prédécesseurs. La résolution de t par m engendre le problème suivant $P'_h = (L, \mathcal{T}, A, M, I, tn' = (T', \prec', \alpha'))$ avec :

  ◇ $T' = (T \setminus \{t\}) \cup T_m$

  ◇ $\prec' = \{(t', t'') \in \prec \mid t' \neq t\} \cup \prec_m$
  $\cup \{(t'', t') \in T_m \times T \mid (t, t') \in \prec\}$

  ◇ $\alpha' = \{(t', \alpha(t')) \mid t' \in T \setminus \{t\}\} \cup \alpha_m$

La résolution d'une tâche abstraite ou primitive d'un réseau de tâches est appelée une *progression*.

L'objectif de CTHD est de trouver un plan *concurrent* dit *en couches* au problème hiérarchique P_h.

Définition 11 *Un plan concurrent en couches* $\Pi = \langle \pi_1, \dots, \pi_n \rangle$ *est une suite de couches d'actions, où chaque couche d'actions est un ensemble d'actions mutuellement indépendantes. La Figure 4.1 représente un tel plan. Plus formellement,* $\forall i \in [\![1, n]\!], \forall (a_1, a_2) \in \pi_i^2$ *, si* $a_1 \neq a_2$*, alors* a_1 *et* a_2 *sont mutuellement indépendantes, autrement dit :*

$$\begin{cases} del(a_1) \cap (pre(a_2) \cup add(a_2)) = \emptyset \\ del(a_2) \cap (pre(a_1) \cup add(a_1)) = \emptyset \end{cases}$$

On dit qu'une couche π_i *est applicable à un état* I *si* $\forall a \in \pi_i$ *,* $pre(a) \subset I$*.*

De plus, on notera pour $i \in [\![1, n]\!]$*,* $\gamma(I, \pi_i)$ *l'état résultant de l'application de la couche* π_i *sur l'état* I*, et si* $\pi_i = \{a_{i,1}, \dots, a_{i,m_i}\}$ *on a :*

$$\gamma(I, \pi_i) = I - \bigcup_{j \in [\![1, m_i]\!]} del(a_{i,j}) \cup \bigcup_{j \in [\![1, m_i]\!]} add(a_{i,j})$$

Enfin, on notera $\gamma(I, [\pi_1, \dots, \pi_i])$ *l'état résultant de l'application successives des couches* $\pi_1, \dots, \pi_i$ *à l'état* I *:*

$$\forall i \in [\![1, n]\!],\ \gamma(I, [\pi_1, \dots, \pi_i]) = \gamma(\gamma(I, [\pi_1, \dots, \pi_{i-1}]), \pi_i)$$

Un plan en couches $\Pi = \langle \pi_1, \dots, \pi_n \rangle$ *est solution du problème* $P_h = (L, \mathcal{T}, A, M, I, tn)$ *si :*

1. Π *est applicable à l'état* I *:* $\forall i \in [\![1, n]\!]$*,* π_i *est applicable à* $\gamma(I, [\pi_1, \dots, \pi_{i-1}])$*.*
2. *il existe un réseau de tâche* $tn_f = (T_f, \prec_f)$ *issu de décompositions successive de* tn *ainsi qu'une bijection* $\sigma : T_f \mapsto \bigcup_{i \in [\![1, n]\!]} \pi_i$ *telle que :*
   - $\forall t \in T_f$*,* $\sigma(t)$ *résout* t
   - $\forall (t_1, t_2) \in (T_f)^2$*, si* $\sigma(t_1) \in \pi_i$ *et* $\sigma(t_2) \in \pi_j$ *et* $t_1 \prec_f t_2$*, alors* $i < j$

Afin de trouver une telle solution en couches, nous encodons le problème hiérarchique en un problème STRIPS. Un problème de planification STRIPS $P_s = (L, A, I, G)$ est également composé de L un ensemble fini de propositions logiques, d'un ensemble fini d'actions A , ainsi que de $I \subset L$ et de $G \subset L$ qui sont respectivement les états initiaux et objectifs du problème.

Nous utilisons la même définition de plan solution que celle exprimée dans la Définition 6, c'est-à-dire une suite d'actions $\Pi = \langle a_1, \dots, a_n \rangle$ telle que $G \subset \gamma(I, \Pi)$. Pour résumer, CTHD encode un problème hiérarchique en un problème STRIPS,

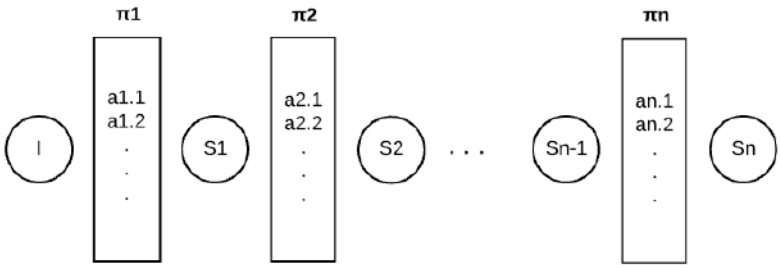

Figure 4.1. : Un plan en couches avec les états successifs résultant de l'application des couches.

et la solution *séquentielle* du problème STRIPS décrit une solution *concurrente* du problème hiérarchique associé.

Pour ce faire, le réseau de tâches initial est encodé dans le problème STRIPS comme un objet. Les actions du problème STRIPS modélisent quant à elles les itérations d'un algorithme permettant de résoudre un problème hiérarchique de manière concurrente. Résoudre le problème STRIPS encodé revient donc à trouver la suite d'opérations algorithmiques à réaliser pour obtenir une solution au problème hiérarchique. L'algorithme en question, appelé **CPFD** (Concurrent Partial-order Forward Decomposition), a été développé à cet effet. CPFD est basé sur l'algorithme de résolution de PFD (Partial-order Forward Decomposition) [GNT04], modifié pour produire des plans concurrents. L'objet de la partie suivante est de décrire le fonctionnement de CPFD.

4.2 CPFD : Un algorithme hiérarchique concurrent

L'algorithme CPFD (voir Algorithme 2) est un algorithme de résolution de problème hiérarchique générant des plans solution en couches. Il est basé sur l'algorithme PFD (voir Algorithme 1).

L'algorithme PFD résout le réseau de tâches et résolvant récursivement les tâches sans prédécesseurs. Les tâches primitives résolues sont ajoutées en séquence au plan solution. À l'inverse, CPFD résout les tâches primitives dans une couche courante, puis ajoute la couche au plan lorsque sa construction est terminée.

CPFD prend quatre paramètres d'entrées : un problème hiérarchique $P = (L, \mathcal{T}, A, M, I, tn)$, un plan en couches Π, un entier i représentant l'indice de la couche actuellement en construction ainsi que τ l'ensemble des tâches primitives de tn choisies pour être résolues dans la couche i. Le principe de l'algorithme est de

construire chaque couche du plan solution successivement. Ainsi, CPFD commence par construire la première couche du plan solution, puis la seconde, etc.

Initialement, le plan en couches Π est initialisé comme vide, l'indice i est initialisé à 0 et $\tau = \emptyset$. À chaque itération, CPFD commence par vérifier s'il reste des tâches à résoudre dans le réseau (ligne 5). Si le réseau est vide, alors Π est renvoyé comme solution du problème hiérarchique. Sinon, une tâche est non déterministiquement choisie parmi les tâches sans prédécesseurs du réseau (lignes 6,7). Un choix non déterministe représente un choix laissé au planificateur : à terme, toutes les options du choix devront être explorées par le planificateur, mais l'ordre dans lequel elles le seront est défini par ce choix. En pratique, ce choix est réalisé au moyen d'une *fonction heuristique*, qui a pour but d'évaluer chaque option. Notons que la forme la plus naïve d'une fonction heuristique est une simple fonction aléatoire. Une fois ce choix non déterministe réalisé, CPFD essaie de résoudre la tâche choisie t du réseau, et sa résolution varie en fonction du type de la tâche sélectionnée :

— **Tâche primitive :** Dans ce cas, les opérateurs pouvant résoudre t sont les actions de A dont (1) les préconditions sont vérifiées dans l'état courant I, et (2) qui sont mutuellement indépendantes avec les actions déjà introduites dans la couche en construction π_i (ligne 8). Ici, plusieurs cas sont possibles :
  — Si une telle action existe, elle est ajoutée à la couche en construction (lignes 19 à 22).
  — Si aucune action n'existe, CPFD différencie deux cas. Soit la couche courante est vide et l'action ne peut pas être résolue du fait que ses préconditions ne sont pas satisfaites. Dans ce cas, CPFD renvoie **Failure** (ligne 10). Soit la couche courante n'est pas vide et l'action peut ne pas pouvoir être résolue à cause des autres actions dans la couche courante. Dans ce cas, la couche courante π_i est appliqué à l'état courant I et CPFD commence la construction de la couche suivante (lignes 11 à 18).
— **Tâche abstraite :** Si la tâche sélectionnée est abstraite, un opérateur m dont les préconditions sont vérifiées dans l'état courant est sélectionné de manière non déterministe (ligne 26). Une nouvelle fois, une heuristique peut être utilisée pour la sélection de l'opérateur. Puis le réseau de tâches tn est décomposé selon la méthode m (lignes 27 à 30). Si aucune méthode n'est disponible, CPFD renvoie **Failure** (ligne 25).

Un exemple d'application de CPFD est représenté dans l'Annexe B.

L'algorithme CPFD a la qualité de préserver les propriétés de **complétude** et de **correction** que possède également PFD [GNT04]. L'exhaustivité représente le fait

Algorithme 1: PFD(P, Π)

```
1  Let P = (L, T, A, M, I, (T, ≺, α))
2  if T = ∅ then return Π
3  tasks ← trail(tn)
4  nondeterministically choose t ∈ tasks
5  if t is primitive then
6      resolvers ← {a ∈ A | task(a) = t, pre(a) ⊆ I}
7      if resolvers = ∅ then
8          return Failure
9      else
10         nondeterministically choose a ∈resolvers
11         I ← γ(I, a)                          // Apply the action effects
12         Π ← Π + [a]                          // Add a to the plan
13         T ← T \ {t}                          // Update the task network
14         ≺' = {(t', t'') ∈ ≺ | t' ≠ t et t'' ≠ t}
15 else
16     resolvers ← {m ∈ M |task(m) = t}
17     if resolvers = ∅ then return Failure
18     nondeterministically choose m ∈ resolvers
19     {m = (T_m, ≺_m)}
20     T ← (T \ {t}) ∪ T_m                      // Decomposing tn with m
21     ≺← {(t', t'') ∈ ≺ | t'' ≠ t} ∪ ≺_m ∪
22         {(t'', t') ∈ T_m × T | (t, t') ∈≺}
23 return PFD(P, Π)
```

d'explorer tous les plans possibles. Cela garantit que si un problème admet au moins une solution, un algorithme exhaustif finira par la trouver. La correction signifie que si une solution est renvoyée par l'algorithme, alors cette solution est une solution correcte (au sens de la définition 11).

Théorème 1 ***CPFD** est **complet** et **correct**.*

Ébauche de preuve (Correction) Tous les plans produits proviennent d'une progression du réseau de tâches initial, il y a donc une séquence de décompositions de tâches produisant les tâches primitives du plan solution. De plus, étant donné qu'une action peut être ajoutée à une couche si la tâche correspondante n'a pas de prédécesseur, toutes les tâches précédant celle ajoutée ont été planifiées sur une couche précédente. Ainsi, les contraintes d'ordonnancement dans $\prec$ sont satisfaites dans le plan solution. Ainsi, les plans renvoyés sont corrects.

Algorithme 2: CPFD(P, Π, i, τ)

Let $P = (L, \mathcal{T}, A, M, I, (T, \prec, \alpha))$
Let π_i the i^{th} layer of Π
Let τ set of tasks already resolved at layer i
if $T = \emptyset$ **then return** Π
tasks $\leftarrow trail(tn) \setminus \tau$
nondeterministically choose $t \in$ tasks
if *t is primitive* **then**
    resolvers $\leftarrow \{a \in A \mid task(a) = t, pre(a) \subseteq I \; and \; (\forall b \in \pi_i, \; a$ independent of $b)\}$
    if resolvers $= \emptyset$ **then**
        if $\pi_i = \emptyset$ **then return** Failure
        else
            $I \leftarrow \gamma(I, \pi_i)$ `// Apply the layer effects`
            $\Pi \leftarrow \Pi + []$ `// Add a new empty layer`
            $i \leftarrow i + 1$ `// Update the layer index`
            $T \leftarrow T \setminus \tau$ `// Update the task network`
            $\prec' = \{(t', t'') \in \prec \mid t' \neq t \; et \; t'' \neq t\} \cup \prec_m \cup$
                $\{(t'', t') \in T_m \times T \mid (t, t') \in \prec\}$
            $\tau \leftarrow \emptyset$ `// Reset the resolved tasks set`
    else
        nondeterministically choose $a \in$ resolvers
        $\pi_i \leftarrow \pi_i \cup \{a\}$ `// Add` a `to the current layer`
        $\tau \leftarrow \tau \cup \{t\}$ `// Add` t `to the resolved tasks`
else
    resolvers $\leftarrow \{m \in M \mid task(m) = t\}$
    if resolvers $= \emptyset$ **then return** Failure
    nondeterministically choose $m \in resolvers$
    $\{m = (T_m, \prec_m)\}$
    $T \leftarrow (T \setminus \{t\}) \cup T_m$ `// Decomposing` tn `with` m
    $\prec \leftarrow \{(t', t'') \in \prec \mid t'' \neq t\} \cup \prec_m \cup$
        $\{(t'', t') \in T_m \times T \mid (t, t') \in \prec\}$
return CPFD(P, Π)

Ébauche de preuve (Complétude) Nous allons démontrer que CPFD est complet en nous basant sur la démonstration que PFD est complet. Soit $P_h = (L, \mathcal{T}, A, M, I, tn)$ un problème HTN et $\Pi = \langle \pi_1, \ldots, \pi_n \rangle$ un plan solution en couches de P_h. Montrons qu'il existe une séquence d'appels récursifs de CPFD produisant Π.

Tout d'abord, notons que pour une couche $\pi = a_1, \ldots, a_k$, toute linéarisation de cette couche $\langle a_{\gamma(1)}, \ldots, a_{\gamma(k)} \rangle$ où γ est une fonction de permutation de $1, \ldots, k$, est une séquence d'actions qui peuvent être appliquées aux mêmes états que π. Cela est dû à la propriété d'*indépendance mutuelle* des actions au sein d'une couche concurrente.

Ainsi, tout plan séquentiel généré en linéarisant toutes les couches de Π (en prenant n'importe quelle fonction de permutation sur chacune de ces couches) est un plan correct qui résout également P. Considérons la linéarisation Π_l définie par les n fonctions de permutation $\gamma_1, \ldots, \gamma_n$.

Puisque PFD est un algorithme complet, il existe une séquence de récursions de PFD qui produit Π_l. À chaque récursion, PFD et CPFD résolvent des tâches sans prédécesseurs, soit abstraites, soit primitives. Tandis qu'ils résolvent les tâches abstraites de la même manière, PFD résout une tâche primitive en ajoutant une action à la fin du plan, tandis que CPFD ajoute l'action résolvant la tâche à la dernière couche du plan. Si la tâche ne peut pas être résolue, PFD renvoie FAILURE, tandis que CPFD tente d'ajouter une nouvelle couche au plan.

Ainsi, pour chaque appel récursif de PFD résolvant une tâche abstraite, l'appel correspondant de CPFD vise à résoudre la même tâche abstraite. Chaque appel récursif de PFD résolvant une tâche primitive est analogue à un appel CPFD ajoutant l'action à la couche actuelle. CPFD insère en plus des opérations de changement d'état entre les appels analogues à ceux de PFD.

4.3 CTHD : Un encodage HTN vers STRIPS concurrent

Dans cette section, nous décrivons le fonctionnement ainsi que l'implémentation de l'encodage CTHD. Par mesure de simplicité, nous faisons trois suppositions (sans perte de généralité) sur les problèmes rencontrés :

1. Le réseau initial de tâches est composé d'une unique tâche dénommée $T0$. Pour les problèmes ayant plusieurs tâches initiales, une tâche et une méthode fictive sont ajoutées. La méthode fictive décompose la tâche fictive en le réseau de tâche initial.
2. Toutes les méthodes du problème hiérarchique admettent une *dernière tâche* dans leur réseau de tâches (une tâche précédée par toutes les autres). Si une méthode n'admet pas une telle tâche, une tâche sans effet ni préconditions est ajoutée à la fin du réseau de tâches.
3. Les méthodes du problème n'ont pas de préconditions. Les méthodes possédant des préconditions sont modifiées, en ajoutant une action qui a pour préconditions les préconditions de la méthode et n'ayant pas d'effets. Cette action est ajoutée au début du réseau de tâches de la méthode, en d'autres termes, elle précède toutes les autres tâches du réseau.

4.3.1 Modélisation du réseau de tâches et de la couche courante

L'idée générale derrière CTHD est d'encoder la méthode de résolution de CPFD en un problème STRIPS. Afin d'encoder l'algorithme CPFD, nous commençons par encoder un réseau de tâches avec STRIPS.

Pour cela nous réutilisons le concept de *taskholders* indroduit dans HTN2STRIPS [Alf+16]. Ces objets STRIPS sont utilisés comme des conteneurs pour les tâches du réseau. Le réseau de tâches est ainsi modélisé par une pile de taskholders et les contraintes entre les tâches sont modélisées par des propriétés booléennes sur ces taskholders. Le nombre de taskholders à disposition définit le nombre maximum de tâches pouvant être contenues dans le réseau. On appelle ce nombre de taskholders la *progression bound*.

Comme cela a été exprimé plus haut, la progression bound a une grande influence à la fois sur le temps de résolution du problème encodé que sur l'existence d'une solution. Si ce nombre est trop petit, la pile de taskholders ne sera pas en mesure de complètement représenter le réseau de tâches et aucune solution ne pourra être trouvée. Si le réseau de tâches est trop grand, les performances de l'encodage en seront altérées.

À l'instar de HTN2STRIPS, CTHD nécessite au moins autant de taskholders qu'il y a de tâches dans le plus grand réseau exploré par progressions successives. À cet effet, HTN2STRIPS '[Alf+16] a proposé une méthode permettant de définir une borne inférieure de la progression bound. Une borne supérieure de la progression bound peut également être définie dans le cas des problèmes hiérarchiques non récursifs. Nous réutilisons ces méthodes d'encadrement de la progression bound pour CTHD également.

Enfin, la couche courante (en construction) est modélisée par un ensemble de propositions. Chaque proposition est associée à une action du problème hiérarchique. Chacune de ces propositions représente le fait que son action associée est présente dans la couche courante.

La Figure 4.2 est une représentation graphique de la structure initiale du problème encodé.

4.3.2 Encodage de la procédure CPFD en actions STRIPS

Afin de modéliser la dynamique de CPFD comme un problème STRIPS, nous définissons trois types d'actions, chacune correspondant à un appel récursif de CPFD :

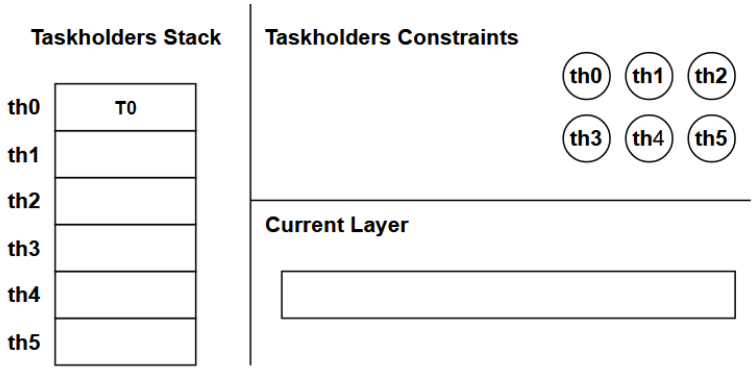

Figure 4.2. : État initial du problème encodé, la tâche initiale est introduite dans le premier taskholder, la couche courante est vide et il n'y a aucune contrainte d'ordre.

— **Actions permettant de résoudre une tâche abstraite :** Ces actions STRIPS représentent la décomposition d'une tâche abstraite par une méthode. Elles s'appliquent à des taskholders n'ayant pas de prédécesseur contenant une tâche abstraite. La tâche abstraite est remplacée par la dernière tâche du réseau de tâches de la méthode et les autres tâches du réseau sont introduites dans des taskholders vides. Un exemple d'application de ce type d'actions est représenté dans la Figure 4.3. Les tâches de la décomposition de $T0$ sont ajoutées à la pile de taskholders et la dernière tâche du réseau remplace $T0$.

— **Actions permettant de résoudre une tâche primitive :** Dans CPFD, la résolution des tâches primitives se fait en les ajoutant à la couche en cours de construction. Dans CTHD, des actions STRIPS analogues sont encodées. Ces actions s'appliquent à un taskholder sans prédécesseurs contenant une tâche primitive. L'action résolvant cette tâche est ajoutée à la couche en contruction si elle est concurrente avec les autres actions de la couche et si ses préconditions sont vérifiées dans l'état courant. Un exemple d'application est représenté dans la Figure 4.4 : le taskholder $th2$ n'a pas de prédécesseurs et contient la tâche primitive $t1$. L'action correspondante $a(t1)$ est ajoutée à la couche courante si ses préconditions sont vérifiées dans l'état courant. Enfin le taskholder $th2$ est marqué comme résolu.

— **Actions permettant de changer de couche :** Ces actions permettent de terminer la couche courante et de commencer la construction de la suivante. La couche courante, ainsi que les taskholders résolus sont vidés et les contraintes les impliquant sont retirées. Un exemple d'application de ce type d'action est représenté dans la Figure 4.5. Dans cet exemple, seule la tâche contenue dans $th2$ a été résolue. Ainsi, les contraintes impliquant $th2$ sont retirées et la couche courante est vidée.

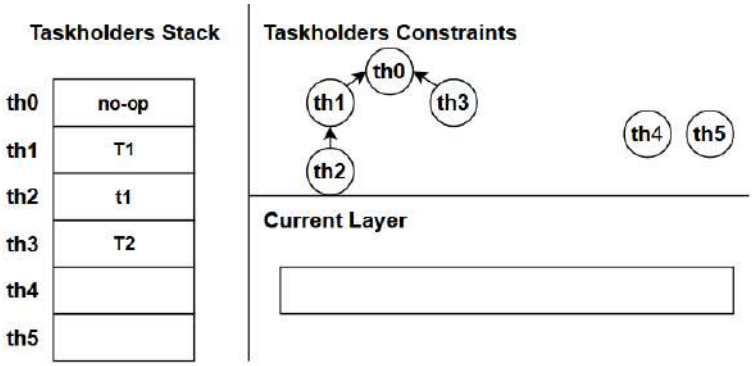

Figure 4.3. : La tâche $T0$ est décomposée en quatre tâches, $T1$ et $T2$ sont abstraites, $t1$ et $no - op$ sont primitives. La tâche $no - op$ représente la dernière tâche du réseau.

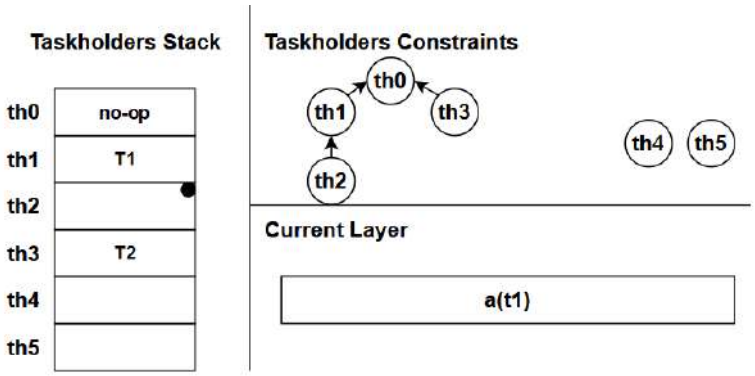

Figure 4.4. : Le taskholder $th2$ n'a pas de prédécesseur et contient une tâche primitive. L'action correspondante $a(t1)$ est ajoutée à la couche courante et le taskholder est marqué comme résolu.

— **Actions permettant de déplacer une tâche d'un taskholder à l'autre :** Ces actions permettent de déplacer une tâche dans un taskholder voisin de celui dans lequel elle est contenue. Ces actions sont appelées des *push actions*, et leur définition a initialement été proposée dans HTN2SAS [Beh+22].

Le planificateur STRIPS, en choisissant quelle action appliquer, choisit finalement quelle progression appliquer au réseau de tâches.

Encodages CTHD (Concurrent TaskHolder Decomposition) :

Nous présentons deux encodages de la procédure CPFD basés sur les concepts présentés précédemment. Ces deux encodages sont respectivement nommés *sCTHD* (Static Concurrent TaskHolder Decomposition) et *pCTHD* (Push Concurrent TaskHolder

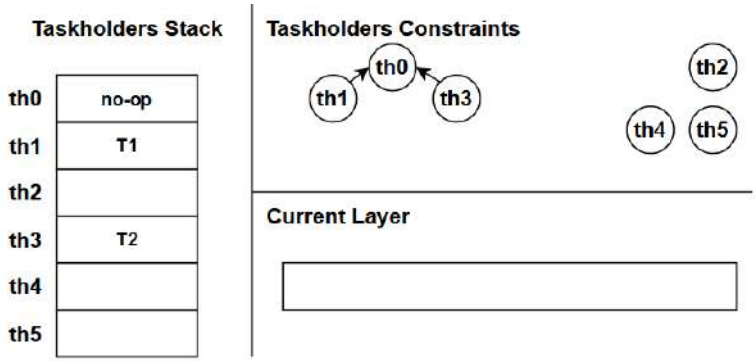

Figure 4.5. : La couche courante est terminée en la vidant et en réinitialisant les taskholders résolus.

Decomposition). La différence entre ces deux encodages porte sur la façon dont les taskholders sont utilisés. Le premier encodage sCTHD utilise les taskholders en utilisant un ordre prédéfini. Le second encodage pCTHD utilise une gestion dynamique de la pile de taskholder en réutilisant le concept de *Push actions* défini par HTN2SAS [Beh+22].

Dans la suite de cette section, nous commençons par présenter les prédicats utilisés pour modéliser le problème STRIPS. Puis, nous présentons les états initiaux et objectifs des problèmes encodés par sCTHD et pCTHD. Enfin, nous présentons les actions STRIPS encodées.

Définition des prédicats :

Nos deux encodages, sCTHD et pCTHD, modélisent la procédure CPFD. À cet égard, ils partagent les prédicats liés à la pile des taskholders, aux contraintes d'ordonnancement et à la modélisation de la couche actuelle. Ils ont également des prédicats spécifiques liés à la gestion de leurs taskholders. Dans la suite, chaque prédicat est utilisé à la fois par sCTHD et pCTHD, sauf indication contraire.

— $(not\_constraint\ ?th1\ ?th2 - taskholder)$ représente les contraintes entre les taskholders. Le prédicat est inversé par mesure de simplicité, ainsi lorsque la proposition $(not\_constraint\ th1\ th2)$ est fausse, cela signifie que la tâche contenue dans $th1$ doit être planifiée avant la tâche contenue dans $th2$.

— $(empty\ ?th - taskholder)$ représente le fait qu'un taskholder soit vide. La proposition $(empty\ th)$ est vraie si le taskholder th ne contient pas de tâche.

— $(in\ ?t - task\ ?th - taskholder)$ est le prédicat représentant si la tâche $?t \in \mathcal{T}$ est contenue dans la taskholder $?th$. La proposition $(in\ t\ th)$ est vraie si t est contenue dans th.

— $(not\_planned\ ?a - action)$ est vrai si a n'est pas dans la couche actuelle du plan.
— $(resolved\ ?th\ -\ taskholder)$ est un prédicat représentant si la tâche contenue dans un taskholder a été résolue dans la couche courante. Ainsi, la proposition $(resolved\ th)$ est vraie si la tâche du taskholder th a été résolue.
— $(prec\_th\ ?th1\ ?th2 - taskholder)$ est un prédicat définissant une relation statique entre les taskholders. La proposition $(prec\_th\ th1\ th2)$ est vraie si $th1$ précède $th2$ dans la pile. Dans la suite, cet ordre sera fixé, et pour tous $0 \leq i, j < b$, la proposition $(prec\_th\ th_i\ th_j)$ sera vraie si et seulement si $i \leq j$. Ce prédicat est spécifique à sCTHD.
— $(next\ ?th\ -\ taskholder)$ est un prédicat utilisé pour indiquer le taskholder priorisé pour la prochaine action *push*. Ce prédicat est spécifique à pCTHD.

Définition des états initiaux et finaux :

L'état initial des problèmes traduits est défini en plaçant la tâche initiale dans le premier taskholder. Les taskholders restants sont définis comme vides et ordonnés dans une pile. Comme il n'y a encore aucune contrainte sur les taskholders, tous les prédicats de contrainte sont initialisés en conséquence.

$I' = I \wedge (in\ task_0\ th_0) \wedge \bigwedge_{1 \leq i < b}(empty\ th_i)$
$\bigwedge_{0 \leq i,j < b}(not\_constraint\ th_i\ th_j) \wedge$
$\bigwedge_{1 \leq i < b}(prec\_th\ th_{i-1}\ th_i) \wedge$
$\bigwedge_{a \in A}(not\_planned\ a)$

Pour l'état final, le problème est résolu lorsque tous les taskholders sont vides, ce qui signifie que toutes les tâches sont planifiées, et lorsque l'état final est atteint. Alors nous avons :

$G' = G \wedge \bigwedge_{i=0}^{p-1}(empty\ th_i)$

Définition des actions sCTHD :

sCTHD génère l'ensemble d'actions $A' = A_c \cup A_p \cup A_l$ où A_c est l'ensemble d'actions résolvant des tâches abstraites, A_p l'ensemble d'actions résolvant des tâches primitives et A_l l'ensemble d'actions de changement de couche :

— *Actions résolvant des tâches abstraites :*
Soit $m = (task(m), pre(m), tn(m))$ et $(task_1, task_2, \ldots, task_k)$ les sous-tâches dans tn. Nous supposons que $task_k$ est la *dernière tâche* de tn. Pour toutes les méthodes $m \in M$, il existe une action $a_m \in A_c$ avec k paramètres $(?th_1, ?th_2, \ldots, ?th_k)$ définis comme suit :

— $pre(a_m) = (in\ task(m)\ ?th_1) \wedge$
$\bigwedge_{i=0}^{b-1}(not\_constraint\ th_i\ ?th_1) \wedge$
$\bigwedge_{i=2}^{k}(prec\_th\ ?th_i\ ?th_{i+1}) \wedge$
$\bigwedge_{i=2}^{k}(empty\ ?th_i)$

— $add(a_m) = \bigwedge_{i=2}^{k}(in\ task_i\ ?th_i) \wedge (in\ task_k\ ?th_1)$

— $del(a_m) = \bigwedge_{i=2}^{k}(empty\ ?th_i) \wedge$
$\bigwedge_{task_i \prec task_j}(not\_constraint\ ?th_i\ ?th_j) \wedge$
$\bigwedge_{i=2}^{k}(not\_constraint\ ?th_i\ ?th_0)$

Notons que les $k-1$ derniers paramètres doivent être des taskholders *ordonnés* selon la relation statique définie par le prédicat $prec\_th$. Par exemple, si trois nouveaux taskholders sont nécessaires pour décomposer la tâche dans $th6$, $(th6\ th2\ th4\ th7)$ est une combinaison valide de paramètres, tandis que $(th6\ th2\ th5\ th3)$ ne l'est pas.

— *Actions résolvant des tâches primitives :*
Pour toutes les actions $a \in A$, il existe une action $a_p \in A_p$ avec un paramètre : un taskholder contenant p et noté $?th$. L'action est définie comme suit :

— $pre(a_p) = pre(a) \wedge (in\ task(a)\ ?th) \wedge$
$\bigwedge_{i=0}^{b-1}(not\_constraint\ th_i\ ?th) \wedge$
$\bigwedge_{t \in nInd(p)}(not\_planned\ ?a)$

— $add(a_p) = add(a) \wedge (empty\ ?th) \wedge (resolved\ ?th)$

— $del(a_p) = del(a) \wedge (not\_planned\ ?a)$

— *Action de changement de couche :*
A_l est composé d'une action conditionnelle a_l sans paramètre. Cette action a une partie non conditionnelle : vider la couche actuelle, et une partie conditionnelle : libérer les taskholders résolus pour une nouvelle utilisation. Elle est définie comme suit :

— $pre(a_l) = \emptyset$

— $add(a_l) = \bigwedge_{a \in A}(not\_planned\ ?a) \wedge$
$\forall(?th\ -\ taskholder)$ when $(resolved\ ?th)$,
$\bigwedge_{i=0}^{b-1}(not\_constraint\ ?th\ \ th_i)$

— $del(a_l) = \emptyset$

Définition des actions pCTHD :

L'encodage pCTHD génère l'ensemble d'actions $A' = A_c \cup A_p \cup A_l \cup A_{Push}$. pCTHD encode la procédure CPFD de la même manière que sCTHD. À cet égard, les ensembles d'actions encodant les *tâches primitives* résolues A_p et l'action de *changement de couche* A_l sont définis de la même manière que dans sCTHD. pCTHD gère

les taskholders comme le fait la version *Push* de HTN2SAS [Beh+22]. Ainsi, nous avons :

— *Actions résolvant des tâches abstraites :*
Soit $m = (task(m), pre(m), tn(m))$ et $(task_1, task_2, \ldots, task_k)$ les sous-tâches dans tn, $task_k$ désigne la *dernière tâche* de tn. Pour chaque méthode $m \in M$, il existe une action $a_m \in A_c$ avec un paramètre $(?th - taskholder)$ défini comme suit :

- $pre(a_m) = (in\ task(m)\ ?th) \wedge$
  $\bigwedge_{i=1}^{b}(not\_constraint\ th_i\ ?th_1) \wedge$
  $\bigwedge_{i=1}^{k-1}(empty\ th_{b-i+1})$
- $add(a_m) = \bigwedge_{i=1}^{k-1}(in\ task_i\ th_{b-i+1}) \wedge (in\ task_k\ ?th)$
- $del(a_m) = \bigwedge_{i=1}^{k-1}(empty\ th_{b-i+1}) \wedge$
  $\bigwedge_{task_i \prec task_j}(not\_constraint\ ?th_{b-i+1}\ th_{b-j+1}) \wedge$
  $\bigwedge_{i=1}^{k-1}(not\_constraint\ th_{b-i+1}\ ?th)$

Dans pCTHD, les sous-tâches héritées d'une décomposition de méthode sont toujours définies dans les mêmes taskholders (les derniers). Cela permet aux actions de méthode de n'avoir qu'un seul paramètre, réduisant ainsi le nombre d'opérateurs générés.

— *Actions Push :*
Pour chaque tâche $t \in \mathcal{T}$, et pour chaque indice de taskholder $2 \leq k \leq b$, il existe une action *Push* $p_t^k \in A_p$ définie comme suit :

- $pre(p_t^k) = (in\ t\ th_k) \wedge$
  $(empty\ th_{k-1}) \wedge$
  $\bigwedge_{i=1, i\neq k}^{b} \neg(next\ th_i)$
- $add(p_t^k) = (in\ t\ th_{k-1}) \wedge (empty\ th_k)$
  $\bigwedge_{i=1}^{b}(not\_constraint\ th_k\ th_i)$
  when $(empty\ th_{k-2}), (next\ th_{k-1})$
- $del(p_t^k) = (next\ th_k) \wedge (empty\ th_{k-1}) \wedge$
  $(in\ t\ th_k) \wedge$
  $\forall(?th - taskholder) :$
  when $\neg(not\_constraint\ ?th\ th_k)$,
  $\bigwedge_{i=0}^{b-1} \neg(not\_constraint\ ?th\ \ th_{k-1})$
  when $\neg(not\_constraint\ th_k\ ?th)$,
  $\bigwedge_{i=0}^{b-1} \neg(not\_constraint\ th_{k-1}\ \ ?th)$

Une action *Push* déplace une tâche et ses contraintes d'un taskholder au taskholder précédent dans la pile. Notons que les actions de *Push* reposent sur des effets conditionnels, permettant un encodage beaucoup plus compact.

4.4 Expérimentations

Dans cette section, nous présentons les expériences et les résultats obtenus pour évaluer l'efficacité de sCTHD et pCTHD. Nous comparons nos encodages avec les deux autres encodages connus : les encodages HTN2STRIPS et HTN2SAS [Alf+16 ; Beh+22]. Notons que ces encodages ne sont pas capables de générer des plans concurrents contrairement à nos encodages. Notre objectif est de montrer comment une procédure concurrente se comporte par rapport à des procédures non concurrentes, tant sur des problèmes avec des plans de solution totalement ordonnés que sur des problèmes admettant des plans concurrents. Nous avons donc sélectionné sept domaines parmi les benchmarks partiellement ordonnés de l'IPC, ainsi qu'un domaine susceptible d'avoir des actions concurrentes, Miconic. Nous avons également ajouté des versions concurrentes de domaines séquentiels où des agents supplémentaires peuvent effectuer les tâches simultanément ($nameConc$ représente la version concurrente du domaine $name$).

4.4.1 Configuration expérimentale

Nous avons exécuté toutes les expériences sur un unique cœur d'un processeur Intel Core i7-9850H en utilisant le planificateur Fast Forward [HN01], implémenté dans la bibliothèque Fast Downward [Hel06]. Toutes les expériences sont limitées à 8 Go de RAM pendant 600 secondes. Nous avons comparé les quatre encodages selon deux critères :

1. **Temps de résolution :** Nous avons mesuré le temps passé par Fast Downward pour trouver une solution aux problèmes traduits. Nous avons effectué une recherche itérative sur la *progression bound* jusqu'à ce qu'une solution soit trouvée.
2. **Qualité du plan :** Comme nous voulons démontrer l'efficacité de notre approche à produire des plans concurrents, nous mesurons le *makespan* (c'est à dire le nombre de couches) de la solution produite par la résolution des problèmes traduits.

Nous avons comparé ces encodages en les évaluant les uns par rapport aux autres. Le score d'un encodage k sur le domaine d avec un ensemble d'instances P_d est défini comme suit :

$$s_d(k) = \frac{1}{|P_d|} \sum_{i \in P_d} \frac{min_{e \in encodages}(score(e,i))}{score(k,i)}$$

Tableau 4.1. : Temps de résolution des encodages

	HTN2STRIPS	HTN2SAS	sCTHD	pCTHD
Satellite	0,20	0,83	**0,95**	0,71
SatelliteConc	0,21	0,79	**0,93**	0,6
Rover	0,02	**1**	0,26	0,85
Transport	0,19	0,58	**0,82**	0,45
Blocksworld	0,29	**0,77**	0,62	0,65
BlowksworldConc	0,34	**0,94**	0,69	**0,94**
Miconic	0,72	**0,84**	0,44	0,82
MiconicConc	0,24	**1**	0,45	0,43
Total	1,56	**6,74**	5,28	5,56

Tableau 4.2. : Makespan des solutions renvoyées

	HTN2STRIPS	HTN2SAS	sCTHD	pCTHD
Satellite	**1**	**1**	**1**	**1**
SatelliteConc	0,94	0,94	**1**	0,95
Rover	0,97	0,97	**1**	0,98
Transport	0,72	0,91	**1**	0,91
Blocksworld	**1**	**1**	**1**	**1**
BlowksworldConc	0,82	0,84	**1**	0,90
Miconic	**1**	**1**	**1**	**1**
MiconicConc	0,76	0,76	**1**	0,91
Total	7,21	7,42	**8**	7,65

où $score(k, i)$ est la mesure de la note de l'encodage k sur l'instance i. Ainsi, chaque encodage est comparé par rapport à celui ayant la meilleure performance sur chacune des instances. Les résultats sont affichés sur les tableaux 4.1 et 4.2.

4.4.2 Commentaires

Temps de Résolution

Le Tableau 4.1 représente les résultats de nos expérimentations en ce qui concerne le *temps de résolution*. Sur les domaines *Satellite* et *Transport*, sCTHD est le plus efficace. Sur les domaines *Rover* et *Blocksworld*, HTN2SAS Push domine les autres encodages. Cette disparité met en évidence la principale différence entre la gestion des taskholders et la gestion de *Push*. Comme les encodages Push décomposent les sous-tâches dans les *derniers* taskholders, ces encodages fonctionnent moins bien sur des problèmes présentant des aspects récursifs tels que Satellite et Transport. De son côté, sCTHD n'est pas affecté par cela et a les mêmes performances quelle que soit la structure du problème. Cependant, les encodages Push sont très efficaces sur des

problèmes avec une faible récursivité, car les actions Push conditionnelles sont alors rarement nécessaires. Dans ce cas les encodages Push représentent le problème de manière très compacte et presque non conditionnelle. Cette efficacité est mise en évidence sur les domaines Rover, Blocksworld et Miconic.

Qualité du Plan

Le Tableau 4.2 représente les résultats concernant le *makespan* des plans de solution. Comme représenté sur ce tableau, sCTHD obtient les meilleurs résultats sur chaque domaine, suivis de l'encodage pCTHD sur chaque domaine également. Cela est dû au fait que ces encodages sont capables de produire des plans concurrents, et donc de réduire le *makespan* de la solution. Notons également que sCTHD et pCTHD peuvent augmenter la concurrence du plan de solution en augmentant le nombre de taskholders. Cela n'est pas visible dans nos expérimentations car les tests se terminent à la première solution valide (donc avec le nombre minimal de taskholders valides). Augmenter le nombre de taskholders dans sCTHD et pCTHD permet cependant de réduire le *makespan* de la solution résultante. Cela est représenté par la Figure 4.6. Il y est représenté le makespan du plan solution trouvé par sCTHD pour un problème de *Satellite* en fonction du nombre de taskholders dans le problème encodé. Comme représenté sur la figure, sCTHD ne peut pas trouver de solution tant qu'il n'atteint pas le nombre minimal de taskholders nécessaire pour former une solution (ici 5). Ensuite, le *makespan* du plan de solution diminue à mesure que le nombre de taskholders augmente. En effet, plus il y a de taskholders disponibles, plus il y a de tâches concurrentes qui peuvent exister simultanément dans le réseau de tâches, et plus de possibilités sont offertes au planificateur STRIPS. Cependant, augmenter le nombre de taskholders a un effet négatif sur le temps de résolution. Plus il y a de taskholders, plus le problème traduit est complexe. Notons que l'augmentation du nombre de taskholders ne peut pas avoir d'impact négatif sur la qualité du plan : plus il y a de taskholders, plus les réseaux de tâches sont accessibles par progression, d'où davantage d'actions concurrentes ajoutées simultanément à la couche actuelle.

Enfin, sur certaines instances, la taille maximale de la progression peut être bornée. L'utilisation de cette borne comme borne de progression amènera sCTHD et pCTHD à explorer et à produire les plans de solution avec le plus petit *makespan*, car chaque réseau de progression peut être représenté dans la pile de taskholders. Une méthode algorithmique pour estimer cette borne a été proposée dans [Alf+16].

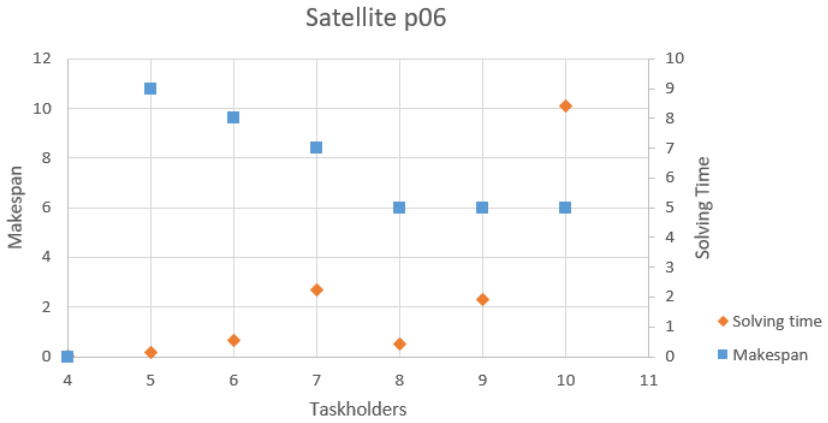

Figure 4.6. : Le *makespan* du plan de solution retourné par sCTHD, ainsi que le temps de résolution, en fonction du nombre de taskholders sur une instance de Satellite.

4.4.3 Discussion sur l'optimalité du makespan dans CTHD

Deux facteurs affectent l'optimalité de l'encodage CTHD : le premier est le nombre de taskholders utilisés dans l'encodage, et le deuxième est le coût associé à chaque action traduite. Avoir plus de taskholders permet à davantage de tâches primitives de coexister simultanément dans la pile de taskholders. Ainsi, il y a plus d'opportunités pour que des tâches concurrentes soient résolues dans la même couche. La solution optimale en termes de *makespan* ne peut être obtenue que si un nombre suffisant de taskholders a été encodé. Le coût des actions traduites permet de guider la procédure de recherche vers une solution de meilleure qualité : trouver la solution optimale en termes de *makespan* signifie trouver le plan avec le plus petit nombre de *couches*. En augmentant le coût des actions de changement de couche, la procédure de recherche est orientée vers de meilleurs plans.

4.5 Conclusion

Dans ce chapitre, nous avons présenté un encodage HTN vers STRIPS appelé CTHD. Cet encodage a la particularité de produire des plans solutions concurrents en couches au problème hiérarchique initial. Nous avons présenté deux versions de cet encodage et comparé ses performances avec les encodages existants dans la littérature, autant quant au temps requis pour trouver une solution qu'à la qualité de la solution (ou son makespan). Ces travaux ont donné lieu à une publication dans une conférence internationale [CPF23a].

L'objet des chapitres suivants sera de permettre la planification de problèmes hiérarchiques temporels, tout en conservant la capacité de produire des plans solutions concurrents.

5 TEP : Un planificateur hiérarchique et temporel

La première contribution se concentrait sur une méthode de planification permettant de proposer un plan concurrent à un problème de planification hiérarchique. L'une des limites de cette contribution est que les actions sont considérées comme instantanées. Cependant, dans un contexte de guerre des mines, la coordination et la synchronisation entre les agents sont des facteurs majeurs de la mission, et la dimension temporelle du problème ne peut pas être ignorée. La deuxième contribution qui a été réalisée au cours de cette thèse porte donc sur un planificateur hiérarchique et temporel dénommé TEP (Temporal Event Planner).

Comme nous avons pu le voir dans la section 3.3.1, il existe trois catégories de problèmes, définies en fonction du degré de concurrence nécessaire dans un plan solution pour ces problèmes. La première catégorie comprend les problèmes *intrinsèquement séquentiels* (toute solution est séquentielle), *mixtes* (admettant des solutions séquentielles et concurrentes) ou *intrinsèquement concurrents* (toute solution est concurrente). L'ambition première du planificateur TEP est de pouvoir résoudre des problèmes hiérarchiques et temporels appartenant à ces trois catégories.

La seconde ambition de TEP est de réutiliser les techniques et heuristiques issues de la planification non temporelle dans le cadre temporel. En effet, comme nous avons pu le voir au terme de l'état de l'art, la majorité des recherches en planification hiérarchique se concentre sur le formalisme non temporel. Ainsi, créer un lien entre planification non temporelle et temporelle permettrait de transférer les avancées du domaine non temporel au domaine temporel. A fortiori, utiliser les heuristiques non temporelles peut entraîner un gain d'efficacité dans le temps de résolution, par rapport aux systèmes de planification purement temporels (comme nous l'illustrerons par la suite).

Pour cela, TEP procède en deux temps. Dans le premier temps, TEP définit un nouveau problème hiérarchique dans lequel les contraintes portant sur la durée des actions sont relaxées. Le problème résultant est suffisamment proche d'un problème non temporel pour appliquer les heuristiques non temporelles. La résolution du problème relaxé renvoie un *squelette* de plan dans lequel les contraintes de durée

sont réinjectées dans un second temps. Puis, TEP cherche à attribuer une date de début pour chaque action du plan satisfaisant les contraintes temporelles en utilisant un solveur CSP (ici ORTools [PDG23]). Un schéma du principe de TEP est représenté dans la Figure 5.1.

Le reste de ce chapitre est organisé de la manière suivante : nous commençons par donner une définition du problème à résoudre, puis nous présentons notre approche **TEP**, et enfin nous présentons les résultats des comparaisons entre TEP et les approches similaires de la littérature.

5.1 Définition du problème

L'ambition de TEP est de résoudre des problèmes hiérarchiques et temporels tels que définis dans HDDL 2.1 [Pel+23]. Ce formalisme est une extension du formalisme hiérarchique défini et utilisé au chapitre précédent. Un problème hiérarchique temporel $P_t = (L, \mathcal{T}, A, M, I, tn)$ est toujours composé d'un ensemble fini de propositions logiques L, d'un ensemble fini de tâches $\mathcal{T}$, d'ensembles finis d'actions A et de méthodes M, d'un état initial $I \subset L$ et d'un réseau hiérarchisé de tâches tn.

À la différence du chapitre précédent, les tâches du problème sont définies de manière temporelle. Cela signifie qu'au lieu d'être réalisée à un unique *instant*, une tâche temporelle est réalisée sur un *intervalle* temporel, lui-même caractérisé par une date de début et une date de fin. Ainsi pour chaque tâche $t \in \mathcal{T}$ on associe deux variables v_t^s et v_t^e représentant respectivement les dates de début et de fin de t. Par la suite on supposera que les tâches sont toutes de durée non nulle et donc que $v_t^s < v_t^e$.

Cette définition temporelle des tâches permet la définition de contraintes d'ordonnancement additionnelles dans le réseau de tâches. Un réseau de tâches temporelles $tn = (T, \prec, \alpha)$ est toujours composé de l'ensemble fini T de symboles de tâches à résoudre, un ensemble de contraintes $\prec$ et $\alpha : T \mapsto \mathcal{T}$ affectant chaque symbole à une tâche de $\mathcal{T}$. La différence repose sur les contraintes contenues dans $\prec$. Dans le contexte temporel, les contraintes portent sur les dates de début et de fin des tâches, et sont caractérisées par les contraintes de Allen [All83] (définies avec les opérateurs $<$, $\leq$ et $=$). L'ensemble des contraintes possibles est représenté sur la Figure 3.10.

Par exemple, si $t_1, t_2 \in T$ alors $v_{t_1}^s \leq v_{t_2}^e$ signifie que le début de la tâche t_1 doit être réalisé avant, ou en même temps, que la fin de t_2. Il faut noter que les contraintes de

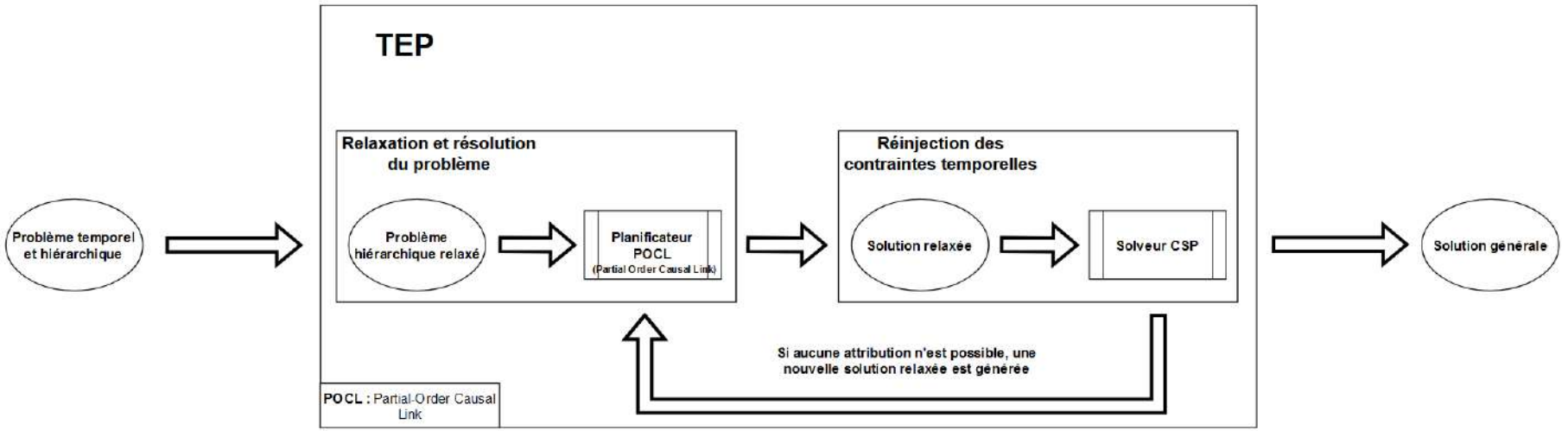

Figure 5.1. : Principe général de TEP, la résolution se déroule en deux temps : un problème relaxé est généré puis résolu, dans un second temps, les contraintes relaxées sont réinjectées et la solution générale est générée en affectant des dates de début aux actions.

type $v^e_{t_1} - v^s_{t_1} = d$ avec $d \in \mathbb{R}$ sont également autorisées pour représenter la durée des tâches primitives.

À ce propos, les tâches primitives sont résolues par des actions temporelles. Les actions temporelles peuvent modifier l'état du problème à deux instants, au début de leur exécution et à la fin. De plus, un ensemble de propriétés doit également être vérifié pendant toute l'exécution de l'action. On appelle cet ensemble les *invariants* de l'action. Plus formellement, nous utilisons la définition des actions temporelles proposée dans PDDL2.1 [FL03].

Définition 12 *Une **action temporelle** $a = (name(a), start(a), end(a), inv(a), d)$ est définie par :*

- $name(a)$ *son nom.*
- $start(a) = (pre_s(a), add_s(a), del_s(a))$ *est une action instantanée au sens de la planification non temporelle. $pre_s(a)$, $add_s(a)$ et $del_s(a)$ sont trois sous-ensembles de L représentant respectivement les préconditions de $start(a)$, les propositions ajoutées après v^s_a et celles retirées.*
- $end(a) = (pre_e(a), add_e(a), del_e(a))$ *est de même une action instantanée au sens non temporel représentant la fin de a.*
- $inv(a) \subset L$ *l'ensemble des invariants de a. Ils doivent être vérifiés sur l'intervalle $]v^s_a, v^e_a[$.*
- $d \in \mathbb{R}$ *la durée de l'action a.*

La Figure 5.2 représente une action temporelle répondant à une telle définition.

On dit que l'action $a = (nom(a), start(a), end(a), inv(a), d)$ est applicable sur un intervalle $[t_1, t_2]$ si :

- $t_2 - t_1 = d$
- $\exists t, t < t_1$ tel que $pre_s(a)$ sont vérifiées sur $]t, t_1[$
- $\exists t', t' < t_2$ tel que $pre_e(a)$ sont vérifiées sur $]t', t_2[$
- $inv(a)$ sont vérifiées sur $]t_1, t_2[$

Une méthode $m = (task(m), pre(m), tn(m)) \in M$ est définie par la tâche $task(m) \in \mathcal{T}$ qu'elle résout, ses préconditions et un réseau de tâches temporelles $tn(m)$.

Supposons maintenant que $tn = (T, \prec, \alpha)$ soit un réseau de tâches temporelles, $m = (task(m), pre(m), tn(m)) \in M$ soit une méthode et que $\exists t \in T$, $\alpha(t) = task(m)$. On note le réseau de tâches $tn(m) = (T_m, \prec_m, \alpha_m)$. La décomposition de tn par m

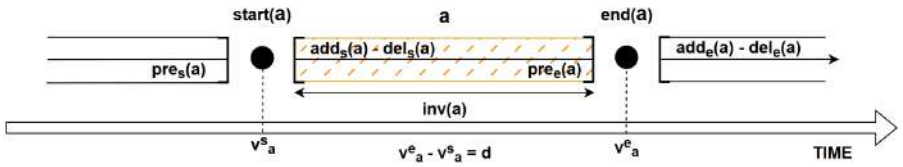

Figure 5.2. : Schéma d'une action temporelle a. Elle est définie sur un intervalle $[v_a^s, v_a^e]$ et est composée de deux actions instantanée $start(a)$ et $end(a)$ ainsi que d'un ensemble d'invariants $inv(a)$ devant être vérifiés sur $]v_a^s, v_a^e[$.

résulte en un réseau de tâches temporelles $tn' = (T', \prec', \alpha')$ défini de la manière suivante :

$$
\begin{aligned}
T' &= (T - \{t\}) \cup T_m \\
\alpha' &= \alpha \cup \alpha_m - \{(t, \alpha(t))\} \\
\prec' = \ & \prec_m \ \cup \{(v_{t_1}^{b_1} < v_{t_2}^{b_2}) \mid \forall (t_1, t_2) \in T^2, (b_1, b_2) \in \{s, e\}^2 \textit{ et } (v_{t_1}^{b_1} < v_{t_2}^{b_2}) \in \prec\} \\
& \cup \{(v_{t_1}^{b_1} \leq v_{t_2}^{b_1}) \mid \forall (t_1, t_2) \in T^2, (b_1, b_2)\{s, e\}^2 \textit{ et } (v_{t_1}^{b_1} \leq v_{t_2}^{b_2}) \in \prec\} \\
& \cup \{(v_{t_1}^{b_1} < v_{t_2}^{b_2}) \mid \forall (t_1, t_2) \in (T - \{t\}) \times T_m, (b_1, b_2) \in \{s, e\}^2 \textit{ et } (v_{t_1}^{b_1} < v_t^{b_2}) \in \prec\} \\
& \cup \{(v_{t_1}^{b_1} \leq v_{t_2}^{b_2}) \mid \forall (t_1, t_2) \in (T - \{t\}) \times T_m, (b_1, b_2) \in \{s, e\}^2 \textit{ et } (v_{t_1}^{b_1} \leq v_t^{b_2}) \in \prec\} \\
& \cup \{(v_{t_1}^{b_1} < v_{t_2}^{b_2}) \mid \forall (t_1, t_2) \in T_m \times (T - \{t\}), (b_1, b_2) \in \{s, e\}^2 \textit{ et } (v_t^{b_1} < v_{t_2}^{b_2}) \in \prec\} \\
& \cup \{(v_{t_1}^{b_1} \leq v_{t_2}^{b_2}) \mid \forall (t_1, t_2) \in T_m \times (T - \{t\}), (b_1, b_2) \in \{s, e\}^2 \textit{ et } (v_t^{b_1} \leq v_{t_2}^{b_2}) \in \prec\} \\
& \cup \{(v_{t_1}^e - v_{t_2}^s = d) \mid \forall (t_1, t_2) \in (T - \{t\})^2 \textit{ et } (v_{t_1}^e - v_{t_2}^s = d) \in \prec\}
\end{aligned}
\tag{5.1}
$$

Enfin, terminons par la définition d'un plan temporel :

Définition 13 *Un **plan temporel** $\pi = \{(a_1, t_1), (a_2, t_2), \ldots, (a_n, t_n)\}$ est un ensemble de couples d'actions temporelles et de dates. Un plan temporel est solution d'un problème temporel $P_t = (L, \mathcal{T}, A, M, I, tn)$ si :*

1. *$\forall i \in [\![1, n]\!]$, a_i est applicable sur $[t_i, t_i + d_{a_i}]$.*
2. *il existe un réseau de tâche temporelles $tn_f = (T_f, \prec_f, \alpha_f)$ issu de décompositions successives de tn ainsi qu'une bijection $\sigma : T_f \mapsto \pi$ telle que :*
   - *$\forall t \in T_f$, $\sigma(t)$ est une action de π résolvant t*
   - *$\forall (t_1, t_2) \in (T_f)^2, (b_1, b_2) \in \{s, e\}^2$, (1) si $(v_{t_1}^{b_1} < v_{t_2}^{b_2}) \in \prec_f$ alors $v_{\sigma(t_1)}^{b_1} < v_{\sigma(t_2)}^{b_2}$ et (2) si $(v_{t_1}^{b_1} \leq v_{t_2}^{b_2}) \in \prec_f$ alors $v_{\sigma(t_1)}^{b_1} \leq v_{\sigma(t_2)}^{b_2}$*

En d'autres termes, π est un plan solution de P_t si π est un plan fixe associé à un réseau de tâches primitives issu de la décomposition de tn.

Dans les sections suivantes, nous détaillons comment TEP parvient à trouver un plan temporel solution d'un problème temporel et hiérarchique.

5.2 TEP : Temporal Event Planning

Afin de résoudre les problèmes temporels, TEP étend la définition de *plan partiel* utilisé dans la planification hybride [BKB14] et dans la planification POCL (Partial Order Causal Link) [MR91]. La planification hybride a été développée dans un contexte de planification non temporelle. Elle a révélé des résultats prometteurs, notamment au travers du planificateur PANDA [Sch15]. Ce planificateur a introduit de nouvelles heuristiques permettant d'améliorer l'exploration de l'espace des plans partiels et ainsi d'accélérer la recherche d'une solution [Ber+17]. Ce type de planification permet de produire des *plans partiellement ordonnés* en résolvant successivement les *défauts* d'un plan partiel initial.

D'autre part, la planification POCL s'adapte particulièrement bien à un contexte de planification temporelle grâce à son concept de *lien causal*. En planification POCL, un *lien causal* est un type supplémentaire de relation, qui relie deux actions non temporelles a_1 et a_2 d'un plan partiel et une propriété p. L'une des deux actions, mettons a_1, produit la proposition p via ses effets et la seconde, a_2, nécessite p comme l'une de ses préconditions. Le lien causal $l = a_1 \xrightarrow{p} a_2$ permet de garantir deux choses : (1) l'action a_1 est ordonnée *strictement avant* l'action a_2 et (2) aucune autre action a_3 du plan partiel qui retirerait la proposition p par ses effets ne peut être ordonnée entre a_1 et a_2. On dit que l supporte la précondition p de a_2, fournie par a_1, et que a_3 est une *menace* de l. Plus simplement, un lien causal permet de garantir que p reste vérifiée sur l'intervalle $]a_1, a_2[$.

L'idée générale de TEP est de représenter un réseau de tâches temporelles comme un plan partiel résultat de la planification type POCL. Pour cela, nous devons légèrement modifier la formalisation d'un plan partiel afin de pouvoir représenter la totalité de l'expressivité de la planification temporelle.

5.2.1 Représentation des réseaux de tâches temporelles

Commençons par définir les plans partiels au sens de la planification POCL (ou hybride). Les plans partiels ont une définition proche de celle d'un réseau de tâches classiques.

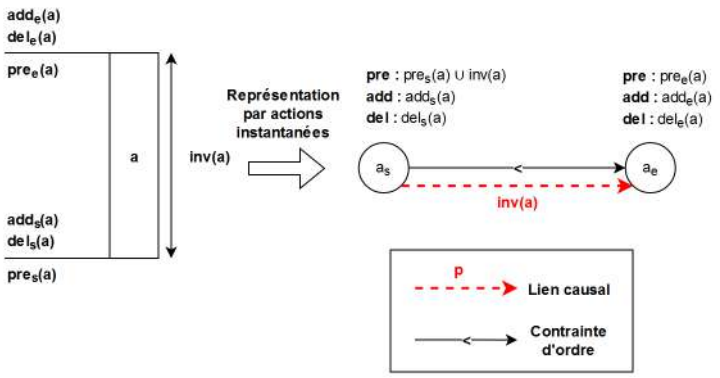

Figure 5.3. : Schéma de la représentation d'une action temporelle par deux actions instantanées et un lien causal.

Définition 14 *Un **plan partiel** $p = (tn, C)$ est composé d'un réseau de tâches (non temporelles) $tn = (T, \prec, \alpha)$ et d'un ensemble de lieux causaux C. De plus, si $l = a_1 \xrightarrow{p} a_2 \in C$ avec $(t_1, t_2) \in T^2$, alors $(t_1 < t_2) \in \prec$ et on a $p \in add(\alpha(t_1))$ et $p \in pre(\alpha(t_2))$.*

*On appelle **défauts** d'un plan partiel soit (1) une tâche abstraite de tn, soit (2) une précondition d'une action du plan partiel qui n'est pas supportée, soit (3) un lien causal est menacé par une action du plan.*

Le principe de résolution des approches POCL est de résoudre successivement les défauts du plan partiel. Si un plan partiel ne contient aucun défaut alors il est une solution au problème de planification.

L'idée principale du planificateur TEP est de relaxer les contraintes de durée du problème, pour ensuite être capable de représenter un réseau de tâches temporelles (relaxées de leur durée) comme un plan partiel de la planification POCL. En effet, une action temporelle peut *presque* être représentée par deux actions *non temporelles* et un *lien causal*. Les actions non temporelles représentent le début et la fin de l'action temporelle, tandis que le lien causal préserve les invariants de l'action temporelle. Un schéma de la représentation des actions temporelles dans TEP est représenté dans la Figure 5.3.

Il faut noter que sur ce schéma, les invariants de l'action temporelle sont traduits par des préconditions de l'action instantanée a_s. Si cela permet de représenter une action temporelle au moyen du formalisme non temporel uniquement, cela ne permet pas de représenter l'entière expressivité de la planification temporelle telle que définie dans HDDL2.1. En effet, considérer les invariants comme des préconditions de a_e signifie que l'action temporelle a a besoin de $inv(a)$ pour démarrer son exécution. Or la planification temporelle est plus subtile que cela, en effet les invariants $inv(a)$

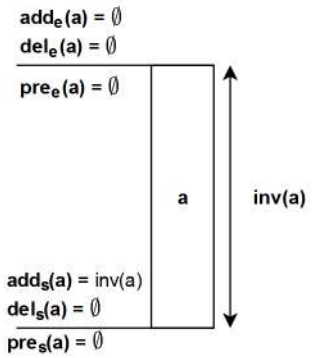

Figure 5.4. : Schéma d'une action non représentable par des actions instantanées classique.

doivent être vérifiés sur l'intervalle *ouvert* $]v_a^s, v_a^e[$. Considérer $inv(a)$ comme des préconditions de a implique que $inv(a)$ doit être vérifié sur l'intervalle $[v_a^s, v_a^e[$, ce qui est plus restrictif. Par exemple, l'action représentée dans la Figure 5.4 satisfait ses propres invariants par ses effets de début et est applicable dans n'importe quel état, ce qui ne serait pas le cas si ces invariants étaient considérés comme des préconditions.

Ainsi, dans le but de représenter complètement l'expressivité de la planification temporelle, nous augmentons dans TEP les actions instantanées de la planification classique et introduisons le concept de *postconditions*. Les *postconditions* sont un ensemble de propriétés qui doivent être vérifiées dans l'état *succédant* à l'application d'une action instantanée. Il y a donc plusieurs cas possibles, soit les postconditions sont déjà vérifiées dans l'état précédant l'action, soit elles doivent être produites par ses propres effets ou ceux d'une autre action planifiée en même instant. Ainsi TEP utilise une définition des actions instantanées intégrant les postconditions, ce nouvelles actions sont appelées des *évènements temporels primitifs* :

Définition 15 *Un* ***évènement temporel primitif*** $e = (nom(e), pre(e), post(e), add(e), del(e))$ *est une extension d'une action instantanée à laquelle est ajoutée un ensemble fini* $post(e)$ *représentant ses postconditions.*

En utilisant ces évènements temporels, il est possible de représenter exactement une action temporelle (dont la durée a été relaxée) comme la combinaison de deux évènements temporels primitifs et d'un lien causal, comme représenté dans la Figure 5.5.

Les tâches abstraites sont quant à elles représentées au moyen de deux *évènements temporels*, l'un abstrait et l'autre primitif, représentant respectivement son début et

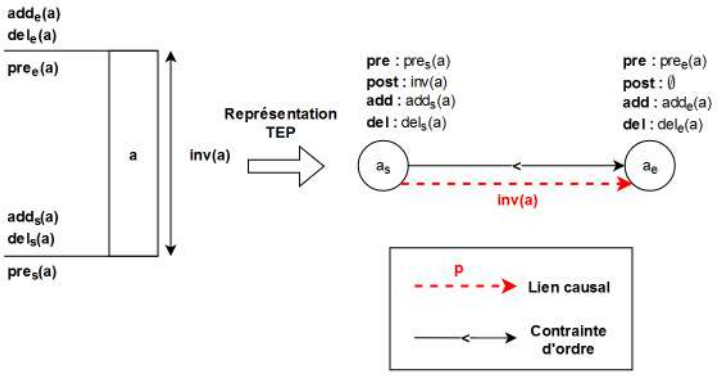

Figure 5.5. : Schéma de la représentation d'une action temporelle dans TEP à l'aide de deux évènements temporels et d'un lien causal.

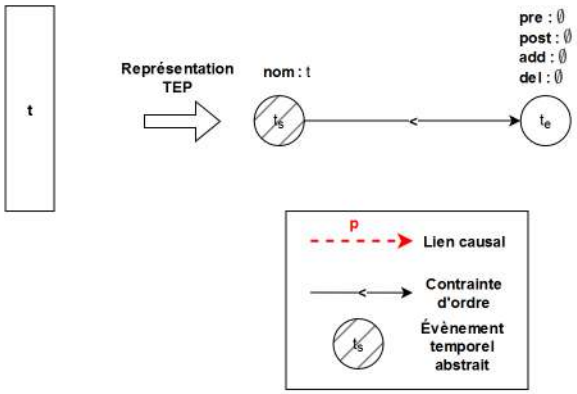

Figure 5.6. : Schéma de la représentation d'une tâche abstraite dans TEP à l'aide de deux évènements temporels, l'un abstrait, l'autre temporel.

sa fin. Un évènement temporel abstrait est défini de manière analogue à une tâche abstraite et a pour vocation d'être décomposé à l'aide d'une méthode. L'évènement primitif est un évènement sans condition ou effet ayant pour vocation de supporter les contraintes portant sur la fin de la tâche abstraite temporelle. La Figure 5.6 montre un schéma de la représentation TEP des tâches abstraites.

Ainsi, on peut définir le *réseau d'évènements temporels* associé à un réseau de tâches temporelles de la manière suivante :

Définition 16 *Soit $tn = (T, \prec, \alpha)$ un réseau de tâches temporelles. Le réseau d'évènements temporels associé à tn est $tn_e = (E, \prec_e, \alpha_e)$ où :*

— *E est l'ensemble des évènements temporels associés à T, E comprend aussi deux évènements spéciaux e_I et e_G représentant respectivement l'état initial et objectif du problème*

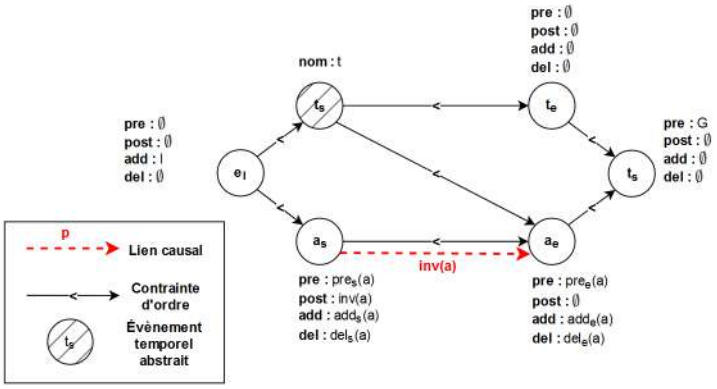

Figure 5.7. : Exemple de représentation d'un réseau de tâches temporelles dans TEP. Deux tâches sont représentées, l'une est une tâche abstraite t et l'autre est primitive a. Le début de t est contraint avant la fin de a.

— *$\prec_e$ est l'ensemble des contraintes analogues à $\prec$, les contraintes sur les débuts et fins de tâches sont reportées sur les évènements temporels associés.*

— *$\alpha_e : E \mapsto \mathcal{T}$ associe à chaque évènement temporel la tâche auquel il est associé.*

De la même façon, un plan partiel d'évènements temporels $\pi_e = (tn_e, C)$ est un couple composé d'un réseau d'évènements temporels et d'un ensemble fini de liens causaux C entre ces évènements temporels.

Un exemple de la représentation d'un plan partiel par TEP est affiché dans la Figure 5.7.

5.2.2 Principe de résolution

Dans cette section nous verrons comment TEP raffine le réseau initial d'évènements temporels pour trouver un plan partiel d'évènements temporels solution. Trois aspects sont présentés, tout d'abord l'algorithme général de résolution est présenté, puis les différents raffinements possibles d'un plan partiel sont détaillés, et enfin les heuristiques utilisées dans TEP sont présentées.

Un algorithme de résolution POCL

Le planificateur TEP repose sur un principe de planification hybride et POCL [BKB14 ; MR91]. À cet effet, l'algorithme de résolution de TEP est fortement inspiré de ces algorithmes. La procédure utilisée dans TEP est représentée sur Algorithme 3.

Algorithme 3: TEP$(L, T, A, M, s_0, w_0, g)$

```
1  π_{e_0} ← the inital plan built from s_0, w_0 and g
2  open ← {π_{e_0}}
3  while open ≠ ∅ do
4      π_e ← heuristically select plan in open
5      open ← open \ {π_e}
6      flaws ← the set of flaws of π_e
7      if flaws = ∅ then
8          V ← search timestamp assignments for π
9          if V ≠ ∅ then
10             return (π_e, V)
11     else
12         φ ← heuristically select a flaw in flaws
13         open ← open ∪ solveFlaw(π_e, φ)
14 return Failure;
```

La procédure TEP prend en entrée un problème hiérarchique et temporel, relaxé de ses contraintes de durée, comme présenté dans les parties précédentes. Cette procédure entremêle deux étapes. Tout d'abord on peut reconnaître la procédure récursive propre aux procédures POCL. Une liste de plans partiels candidats est initialisée avec le plan partiel associé au réseau de tâches temporelles initial (lignes 1 et 2). Puis, à chaque itération TEP sélectionne l'un des plans candidats au moyen d'une première fonction heuristique appelée l'*heuristique de sélection de plan* (ligne 4). Puis, TEP sélectionne l'un des défauts de ce plan et le résout (lignes 12 et 13). La sélection se fait au moyen d'une seconde heuristique, l'*heuristique de sélection de défauts*. Les nouveaux plans générés sont rajoutés à la liste des plans candidats. Si un plan sans défaut est sélectionné, TEP cherche à affecter des dates aux événements temporels du plan partiel (ligne 8). C'est au moment de cette affectation que les contraintes de durée sont réinjectées dans le problème. Si une affectation est trouvée, la solution est renvoyée comme une solution au problème général (ligne 10). Sinon, TEP renvoie un échec (ou Failure, ligne 14).

La particularité de TEP est d'ajouter une étape d'affectation après la phase de planification POCL. Le problème n'est considéré comme réellement temporel qu'à ce moment là.

Défauts : définition et résolution

Les défauts rencontrés dans un plan partiel d'évènements temporels sont très similaires à ceux qui sont rencontrés dans un plan partiel de planification POCL

classique. Cependant, l'expressivité des problèmes temporels entraîne de légères modifications quant à la résolution des défauts par rapport aux planificateurs POCL. Nous comptons tout de même trois types de défauts, listés ci-dessous :

— **Défauts de condition ouverte** : Un plan partiel a un défaut de condition ouverte pour chaque *précondition* ou *postcondition* p d'un évènement temporel e_1 n'étant pas supporté par un lien causal. La résolution se fait en ajoutant un *lien causal* $l = e_2 \xrightarrow{p} e_1$ entre un évènement temporel e_2 ayant p dans ses effets et e_1. Une contrainte d'ordre est également ajoutée : $(e_2 < e_1)$ si la condition est une précondition et $(e_2 \leq e_1)$ dans le cas d'une postcondition.

— **Défauts de menace causale** : Supposons qu'une tâche t_k menace un lien causal $\langle t_i \xrightarrow{p} t_j \rangle$. Cette menace peut être résolue de deux manières différentes. La première consiste à contraindre strictement t_k avant la tâche t_i qui produit p en ajoutant $(t_k < t_i)$ dans les contraintes d'ordonnancement. La seconde consiste à contraindre t_k après ou en même temps que la tâche t_j en ajoutant la contrainte $(t_j \leq t_k)$. Comme toujours, l'ajout de contraintes d'ordonnancement à π doit garantir la cohérence des contraintes. La réparation d'une menace est très similaire à la réparation des menaces dans la planification hybride, sauf que nous autorisons des contraintes d'ordonnancement non strictes.

— **Défauts de décomposition** : Il y a un défaut de décomposition pour chaque évènement temporel abstrait encore dans le plan partiel. La résolution de ces défauts se fait en remplaçant l'évènement abstrait par le plan partiel associé au réseau de tâches de la méthode choisie pour la décomposition.

Il faut noter qu'il peut y avoir plusieurs façons de résoudre un même défaut. Dans le cas des défauts de conditions ouvertes, il y a autant de résolutions possibles qu'il y a d'évènements temporels ayant la condition en question pour effet. Dans le cas des défauts de décomposition, il y a autant de résolutions qu'il y a de méthodes associées à l'évènement temporel abstrait. Enfin, les défauts de menaces causales ont toujours deux résolutions, comme décrit plus haut. Dans l'Algorithme 3, le rôle de la fonction $solveFlaw(\pi, \phi)$ est de renvoyer l'ensemble des plans partiels issus des différentes résolutions possibles du défaut ϕ de $\pi = (tn, C)$, avec $tn = (E, \prec, \alpha)$. L'Algorithme 4 décrit le fonctionnement de $solveFlaw$.

Des exemples de résolutions de défauts sont illustrés sur les Figures 5.8, 5.9, 5.10, 5.11 et 5.12. La Figure 5.8 représente le réseau d'évènements temporels initial que nous considérons ici. Il est composé de huit évènements temporels, dont un abstrait. Une précondition est à satisfaire pour l'évènement a_e. Les évènements a_s et b_s ont chacun une postcondition à satisfaire. Aussi, le lien causal $a_s \xrightarrow{p} a_e$ est menacé par l'évènement b_e.

Algorithme 4: solveFlaw((π_e, ϕ))

```
1  res ← ∅
2  if φ is an Open condition flaw then
3      p ← the condition to fulfill
4      Producers ← {e ∈ E | p ∈ add(e)}
5      for prod in Producers do
6          if p represents a precondition then
7              ≺' ← ≺ ∪ (prod < e_1)
8          else
9              ≺' ← ≺ ∪ (prod ≤ e_1)
10         tn' ← (E, ≺', α)
11         C' = C ∪ ⟨prod --p--> e_1⟩
12         π' ← (tn', C')
13         res ← res ∪ {π'}
14 if φ is a Decomposition flaw then
15     t ← the task to decompose
16     M_t ← {m ∈ M | task(m) = t}
17     for m in M_t do
18         tn' ← the task network resulting from the decomposition of t by m
19         π' ← (tn', C)
20         res ← res ∪ {π'}
21 if φ is a Causal threat flaw then
22     ⟨e_1 --p--> e_2⟩ ← the threatened causal link
23     threat ← the threatening task
24     tn_1 ← (E, ≺ ∪ (threat < e_1), α)
25     tn_2 ← (E, ≺ ∪ (e_2 ≤ threat), α)
26     res ← res ∪ {(tn_1, C), (tn_2, C)}
27 return res;
```

La Figure 5.9 représente le réseau résultant de la résolution du défaut de tâche abstraite et donc de la décomposition de t en deux sous-tâches abstraites $t1$ et $t2$.

La Figure 5.10 montre la résolution de la précondition ouverte de a_e en ajoutant un lien causal avec un autre évènement produisant cette propriété, b_s.

Enfin, les Figures 5.11 et 5.12 représentent les deux façons de résoudre le défaut de menace causale entre $a_s \xrightarrow{p} a_e$ et b_e.

Heuristiques de recherche

Le principal avantage de TEP est d'exploiter (avec quelques adaptations présentées ci-dessous) les heuristiques développées pour la planification hybride non temporelle

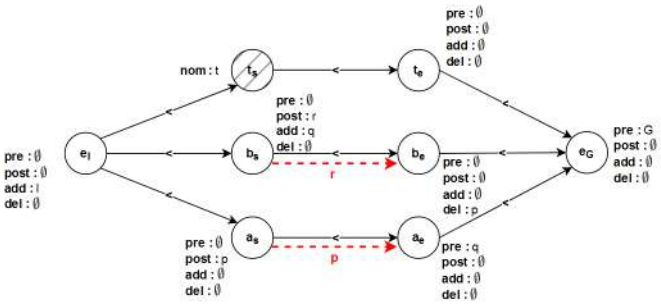

Figure 5.8. : À l'état initial, trois tâches sont représentées dans le réseau, soit huit évènements au total. e_I et e_G représentent les états initiaux et finaux, a_s et a_e sont les évènements à a, b_s et b_e à b et t_s et t_e à la tâche abstraite t.

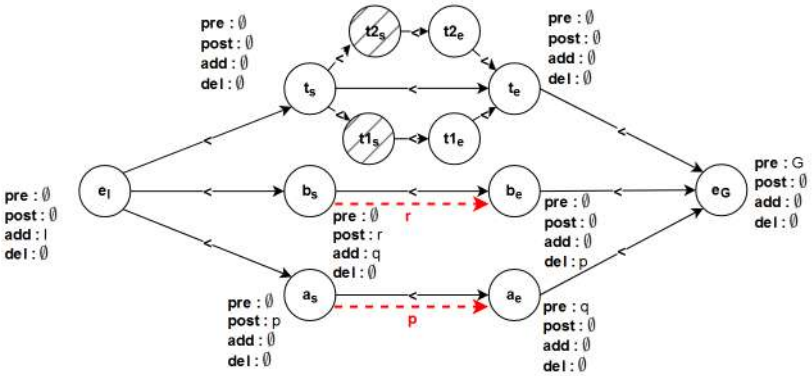

Figure 5.9. : La tâche abstraite est décomposée en insérant le réseau d'évènements temporels associé à la méthode utilisée dans la décomposition.

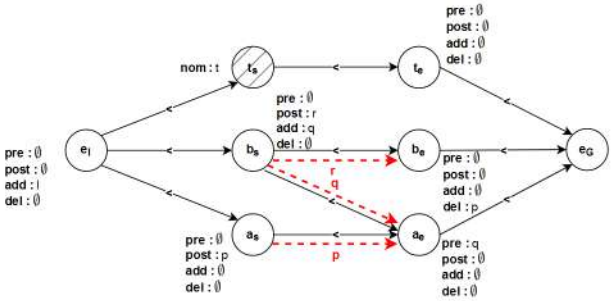

Figure 5.10. : Un défaut de précondition ouverte est résolu en ajoutant un lien causal couplé à une relation d'ordre avec un évènement produisant la propriété requise.

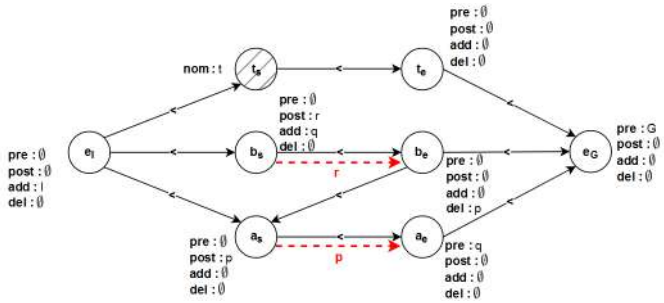

Figure 5.11. : Première manière de résoudre une menace causale, l'évènement menaçant le lien causal est ordonné strictement avant le lien causal.

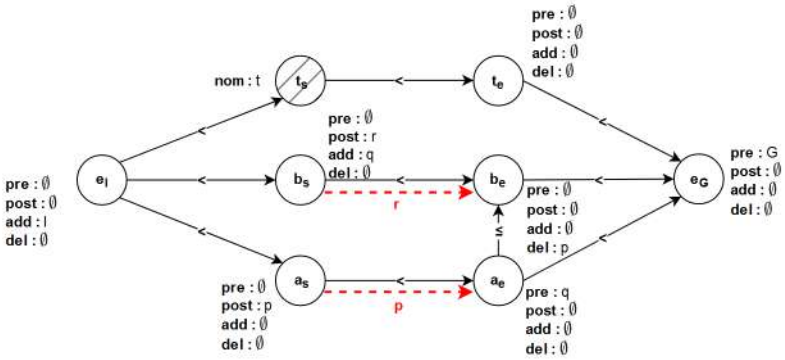

Figure 5.12. : Seconde manière de résoudre une menace causale, le lien causal est ordonné de telle sorte à se terminer avant que, ou en même temps que, se présente la menace.

afin de résoudre des problèmes temporels. La procédure de recherche TEP (voir Algo. 3) repose sur deux choix, qui sont guidés par deux fonctions heuristiques. La première effectue une sélection non déterministe parmi l'ensemble des plans partiels candidats et décide lequel explorer en premier (ligne 4). Les fonctions heuristiques guidant ce choix sont appelées *heuristiques de sélection de plan* et ont un impact significatif sur les performances de la recherche (en d'autres termes, le temps nécessaire pour trouver un plan de solution) et sur la qualité du plan retourné (c'est-à-dire la durée totale du plan solution).

Le deuxième choix (ligne 9) effectue une sélection parmi les défauts à résoudre guidé par l'heuristique de sélection de défauts. Les fonctions heuristiques guidant ce choix sont appelées *heuristiques de sélection de défaut*. Notez que chaque défaut dans le plan partiel doit être résolu afin de trouver un plan solution. Ainsi, la *complétude* de la procédure TEP dépend uniquement des heuristiques de sélection de plan. Cependant, l'*ordre* dans lequel les défauts sont résolus a un impact significatif sur les performances de la procédure de recherche. Les heuristiques de sélection de défauts jouent donc un rôle dans l'efficacité de la procédure mais par sur sa complétude. Dans ce qui suit, nous présentons les heuristiques de sélection de plan et de défaut telles qu'implémentées dans TPE.

Heuristiques de sélection de plan Parmi les heuristiques de sélection de plan en planification hybride non temporelle, par exemple, [SBB07 ; Sch09 ; BKB14], les plus efficaces sont celles développées par PANDA [BKB14]. Leur principe est d'estimer le nombre de raffinements nécessaires pour atteindre un plan solution à partir du plan partiel actuel. L'idée de PANDA était d'étendre ce principe à la planification hybride en estimant de manière plus précise le nombre de défauts dans un plan en tenant compte des défauts qui seront introduits par des tâches qui n'ont pas encore été décomposées. Cette estimation repose sur une structure appelée Task Decomposition Graph (TDG) qui encode la décomposition du problème hiérarchique.

Dans la section suivante, nous présentons (1) comment TEP étend le concept de TDG pour encoder non seulement les décompositions de tâches abstraites mais aussi les décompositions de tâches temporelles dans une nouvelle structure appelée Temporal Task Decomposition Graph (TTDG), et (2) comment les heuristiques classiques pour la planification hybride développées dans [BKB14 ; Ber+17] peuvent être dérivées à partir d'un TTDG pour résoudre des problèmes de planification hybride temporelle.

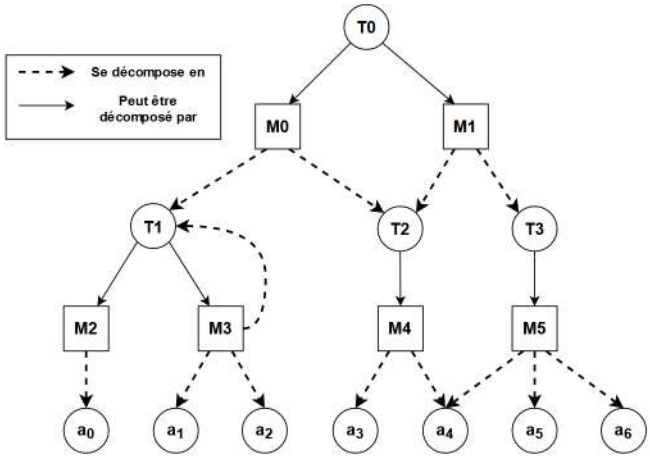

Figure 5.13. : Un exemple de TTDG, la tâche initiale $T0$ peut être décomposée par deux méthodes $M0$ et $M1$. Les nœuds représentant les méthodes sont eux-mêmes reliés aux tâches introduites par l'application de la méthode.

Définition 17 *Un TTDG d'un problème de planification $P = (L, \mathcal{T}, A, M, I, tn)$ est un graphe bipartite orienté ET/OU $G = (V_T, V_M, E_{T,M}, E_{M,T})$, où V_T est un ensemble de sommets comprenant des actions temporelles et des tâches abstraites qui peuvent être obtenues en décomposant le plan partiel initial π_0 construit à partir de tn. V_M est un ensemble de sommets de méthodes qui décomposent une tâche abstraite dans V_T. Puis, $E_{T,M}$ est un ensemble d'arêtes qui relient les nœuds de V_T à V_M. Chacune des arêtes de $E_{T,M}$ relie une tâche abstraite à une méthode qui la décompose. Enfin, $E_{M,T}$ est un ensemble d'arêtes qui relient les nœuds de V_M à ceux de V_T. Chacune des arêtes de $E_{M,T}$ relie une méthode à une tâche de sa décomposition.*

Pour des raisons de brièveté, les nœuds enfants d'un nœud $v \in V_T \cup V_M$ sont notés $child(v) = \{v_i | (v, v_i) \in E\}$ avec $E = E_{T,M} \cup E_{M,T}$. Un TTDG est illustré dans la Figure 5.13. Dans le cas général, un TTDG comme un TDG peut être un graphe cyclique.

Pour généraliser les différentes heuristiques de sélection de plan développées dans la planification hybride à la planification temporelle, nous devons d'abord définir deux estimations qui peuvent être extraites d'un TTDG : (1) une estimation du nombre de décompositions nécessaires pour décomposer le plan en tâches primitives, et (2) une estimation du nombre de conditions ouvertes à satisfaire.

Ces deux estimations reposent sur les concepts de tâches obligatoires et de cardinalité des tâches. Les tâches obligatoires, notées $M(t)$, sont des tâches concrètes qui apparaissent dans toutes les méthodes de décomposition ou actions duratives associées à la même tâche t. Plus simplement, l'ensemble des tâches obligatoires $M(t)$ comprend les tâches qui seront nécessairement incluses dans un plan partiel

lorsque la tâche t est décomposée. Par exemple (voir Figure 5.13), nous avons $M(T0) = \{T2, a_3, a_4\}$, $M(T1) = \{a_0\}$, $M(T2) = \{a_3, a_4\}$ et $M(T3) = \{a_4, a_5, a_6\}$. La cardinalité de la tâche (TC) d'une tâche t peut être définie de manière directe comme le nombre de tâches dans son ensemble obligatoire, c'est-à-dire $TC = |M(t)|$. Cette cardinalité de tâche sert de borne inférieure pour estimer l'effort nécessaire pour décomposer la tâche t.

Le calcul de la première estimation correspond au calcul de TC. En pratique, il est calculé de la manière suivante :

$$TC(v) = \begin{cases} 1 & \text{si } v \in V_T \text{ et } v \text{ est primitive} \\ \min\limits_{v_i \in child(v)} TC(v_i) & \text{si } v \in V_T \text{ et } v \text{ est abstraite} \\ \sum\limits_{v_i \in child(v)} TC(v_i) & \text{si } v \in V_M \end{cases}$$

La deuxième estimation, connue sous le nom de *MME* (Minimal Modification Effort), représente le nombre de conditions ouvertes, comprenant à la fois les préconditions et les postconditions, qui doivent être satisfaites pour chaque nœud. Elle peut être déterminée à partir d'un TTDG comme suit :

$$MME(v) = \begin{cases} 1 + |pre(v) \cup post(v)| & \text{si } v \in V_T \text{ et } v \text{ est primitive} \\ \sum\limits_{d \in child} TC(d) & \text{si } v \in V_M. \\ 1 + \min\limits_{d \in child} TC(d) & \text{si } v \in V_T \text{ et } v \text{ est abstraite} \end{cases}$$

où $pre(v)$ représente les préconditions d'un nœud primitif $post(v)$ ses postconditions.

Enfin, nous étendons les quatre heuristiques classiques proposées dans [BKB14; Ber+17] aux plans temporels $\pi = (tn_e, C)$ de la manière suivante. Ici, $tn_e = (E, \prec_e, \alpha_e)$ est un réseau d'évènements temporels, tandis que C représente les liens de causalité de π, et F désigne ses défauts.

$$\begin{aligned} h_{TC}(\pi) &= \sum_{i \in I, \alpha(i) \text{ est abstrait}} TC(\alpha(i)) \\ h_{MME}(\pi) &= \sum_{i \in I} MME(\alpha(i)) \\ h_{TDGm}(\pi) &= h_{MME}(\pi) - |C| \\ h_{TC\,+\,\#F}(\pi) &= h_{TC}(\pi) + |F| \end{aligned}$$

Il faut remarquer que toutes ces heuristiques sont *admissibles* par rapport aux nombres de modifications introduites dans le plan partiel [Ber+17]. Cependant, elles ne permettent pas de garantir l'optimalité du plan solution par rapport à la durée totale de la mission.

Heuristiques de sélection de défaut Les heuristiques de sélection des défauts n'impactent pas la complétude de la procédure TEP. Cependant, elles influent grandement sur l'efficacité globale de la recherche. En particulier, détecter si un défaut n'a pas de résolution possible peut économiser beaucoup de temps, en éliminant du même coup une partie de l'espace de recherche.

La stratégie la plus courante pour sélectionner l'ordre des défauts est de choisir en premier lieu ceux avec le moins de résolutions possible. En d'autres termes, les défauts les plus contraints. Cette stratégie est connue sous le nom de LCFR (Least Cost Flaw Repair) [JP94]. Bien que cette heuristique de sélection de défauts ait été initialement développée pour des problèmes non hiérarchiques, elle a été utilisée en planification hybride par le planificateur hybride PANDA [BKB14]. Lors de son adaptation à la planification hiérarchique, la priorité est donnée à la décomposition des tâches abstraites par rapport aux autres défauts afin de maintenir l'exhaustivité de la procédure. Cela est dû au fait que le nombre de résolutions pour un défaut dépend de la décomposition des autres tâches.

TEP met en œuvre la même stratégie de sélection de défauts que PANDA, en donnant la priorité aux défauts de décomposition, puis en priorisant les défauts restants en commençant par les plus contraints.

5.3 Expérimentations

Dans cette section, nous comparons TEP et ses heuristiques avec l'approche *Timeline* la plus efficace proposée par FAPE [Bit+20]. L'approche *timeline* est actuellement la seule approche pour résoudre les problèmes temporels hiérarchiques dans la troisième catégorie de Cushing. Ainsi, seule les approches *timeline* partagent l'expressivité de TEP.

5.3.1 Configuration expérimentale

Les deux planificateurs, TEP et FAPE, ont été testés sur un seul cœur d'un processeur Intel Core i7-9850H, avec une limite de 8 Go de RAM et une limite de temps de 600 secondes. Pour assurer une comparaison équitable, les deux planificateurs ont été implémentés en utilisant la même bibliothèque PDDL4J [PF18], et ont utilisé le même solveur CSP, celui fourni par OR-Tools [PDG23].

Les critères d'évaluation sont (1) le *temps de résolution*, qui représente le temps nécessaire pour résoudre un problème, (2) le *makespan* d'un plan solution, qui représente la durée totale du plan, et enfin, (3) la *couverture* de chaque planificateur, c'est-à-dire le nombre de problèmes résolus de chaque domaine. À noter que le temps de résolution et le makespan sont présentés sous forme de scores IPC.

Le score IPC permet d'attribuer un score en 0 et 1 à chaque planificateur et pour chaque problème benchmark. La particularité du score IPC est que le score attribué à un planificateur pour un exemple dépend des résultats obtenues par les autres planificateurs sur cet exemple. Pour un ensemble de planificateurs $\{P_1, \ldots, P_n\}$, un problème de planification benchmark p, si $\{C(P_1, p), \ldots, C(P_n, p)\}$ sont les valeurs obtenues respectivement par les planificateurs sur le problème p selon un critère d'évaluation C à minimiser, alors la score IPC obtenu par chacun des planificateurs sur ce problème est le suivant :

$$\forall i \in [\![1, n]\!], IPC(P_i, p) = \frac{\min_{j \in [\![1,n]\!]} C(P_j, p)}{C(P_i, p)}$$

Ainsi, le meilleur planificateur sur un problème obtiendra le score de 1, qui est le score maximum. Le score total d'un planificateur est ensuite la somme des scores qu'il a obtenu sur chacun des exemples.

5.3.2 Benchmarks de planification

Il n'existe actuellement aucun benchmark standard disponible pour la planification hiérarchique et temporelle. Par conséquent, nous avons proposé un ensemble de benchmarks basés sur le langage HDDL2.1 [Pel+23], qui vise à faciliter la comparaison des planificateurs hiérarchiques et temporels. Parmi ces benchmarks, certains sont des adaptations de domaines hiérarchiques non temporels des compétitions de planification IPC. Ils ont été modifiés en ajoutant des durées aux actions : *Gripper*, *Satellite* et *Rover*.

Dans le domaine *Gripper*, toutes les solutions sont séquentielles, et aucune concurrence temporelle n'est requise. Il s'agit donc d'un domaine de première catégorie de Cushing. Dans les domaines *Satellite* et *Rover*, des problèmes avec plusieurs satellites ou rovers permettent la concurrence, mais elle n'est pas obligatoire. Le planificateur peut choisir de trouver soit un plan simple et non concurrent, soit un plan concurrent en utilisant plusieurs rovers. Ces problèmes appartiennent donc à la deuxième catégorie de Cushing.

De plus, certains domaines ont été spécifiquement conçus pour correspondre à la troisième catégorie de Cushing :

— *Area Scan* modélise un ensemble de dispositifs hétérogènes qui coopèrent pour scanner des zones. Area Scan a deux versions : une version totalement ordonnée, où les tâches abstraites des problèmes sont totalement ordonnées, et une version désordonnée où la concurrence peut également être obtenue grâce à des tâches abstraites concurrentes.

— *Table Carriers* s'inspire du domaine *Gripper*. Dans ce domaine, les agents doivent coopérer pour transporter des tables, les tables nécessitant une, deux ou trois personnes agissent simultanément pour les transporter.

5.3.3 Résultats

Les résultats sont présentés dans le Tableau 5.1. En termes de critère de *temps de résolution*, TEP surpasse FAPE sur tous les domaines. Cependant, la performance des heuristiques varie d'un domaine à l'autre. Pour les domaines *Gripper* et *Satellite*, TEP couplé à MME, dénommé par $TEP(MME)$, a obtenu les meilleurs scores. En revanche, pour le domaine *Rover*, la configuration $TEP(TDGm)$ obtient le meilleur score pour la métrique du temps de résolution. Cette disparité peut être attribuée aux différences entre MME et $TDGm$.

Contrairement à MME, qui mesure uniquement les propositions à satisfaire au sein d'un plan partiel, l'heuristique $TDGm$ tient également compte du nombre de liens causaux déjà intégrés dans le plan partiel. Ainsi, $TDGm$ incite à raffiner continuellement le même plan partiel en lui ajoutant des liens causaux jusqu'à ce qu'il forme une solution, ou soit éliminé. Cette heuristique a donc un comportement en profondeur d'abord. L'heuristique MME à l'inverse favorise un comportement de largeur d'abord.

Des différences similaires peuvent être observées entre TC et $TC + \#F$. Alors que la configuration $TEP(TC)$ obtient des scores supérieurs dans le domaine *AreaScan PO*,

	Gripper			**Satellite**			**Rover**			**AreaScan PO**			**AreaScan TO**			**TableCarriers**			**Total**		
	S. Time	Span	Cov.	S. Time	Span	Cov.	S. Time	Span	Cov.	S. Time	Span	Cov.	S. Time	Span	Cov.	S. Time	Span	Cov.	S. Time	Span	Cov.
TEP(TC)	6.72	**7.00**	**7/10**	6.21	6.50	**8/9**	4.96	5.43	**10/10**	19.99	**21.83**	**22/22**	19.93	**21.97**	**22/22**	7.13	8.33	9/10	64.94	71.05	**78/83**
TEP(MME)	**6.81**	**7.00**	**7/10**	**7.09**	6.71	**8/9**	6.62	7.41	9/10	17.64	21.62	**22/22**	20.25	21.74	**22/22**	**8.57**	9.34	**10/10**	**66.99**	73.82	**78/83**
TEP(TDGm)	4.09	6.00	6/10	5.24	7.38	**8/9**	**8.56**	7.63	9/10	18.67	21.62	**22/22**	**21.71**	21.74	**22/22**	4.64	**9.77**	**10/10**	62.90	**74.14**	77/83
TEP(TC + #F)	4.79	6.00	6/10	6.71	**7.63**	**8/9**	6.67	7.26	**10/10**	**20.39**	21.62	**22/22**	21.48	21.77	**22/22**	3.67	8.61	9/10	63.70	72.88	77/83
FAPE	1.52	6.00	6/10	2.74	5.88	6/9	3.64	**7.99**	8/10	10.00	18.92	19/22	13.69	21.72	**22/22**	2.73	7.00	7/10	34.31	67.51	68/83

Tableau 5.1. : Résultats des expérimentations pour les différentes configurations.

$TEP(TC + \#F)$ est la configuration la plus performante pour le domaine *AreaScan TO*. Encore une fois, l'incorporation du nombre de défauts restants dans $TC + \#F$ incite une stratégie de recherche en profondeur d'abord, ce qui est avantageux pour certains domaines. En revanche, le comportement en largeur d'abord de l'heuristique TC conduit à de meilleures performances dans d'autres domaines.

D'autres configurations donnent les meilleurs scores pour la métrique du *makespan*. Pour les domaines *Gripper* et *AreaScan*, $TEP(TC)$ émerge comme la configuration la plus performante. En revanche, dans le domaine *Satellite*, $TEP(TC + \#F)$ présente la meilleure performance, tandis que dans le domaine *TableCarriers*, $TEP(TDGm)$ surpasse les autres. De plus, FAPE obtient le meilleur score pour le domaine *Rover*. Dans l'ensemble, les configurations obtiennent des résultats comparables pour les domaines testés. Enfin, en termes de couverture, TEP surpasse FAPE dans tous les domaines.

5.4 Conclusion

Nous avons présenté TEP, une approche permettant de résoudre des problèmes à la fois hiérarchiques et temporels. Elle consiste à relaxer les problèmes temporels en problèmes non temporels afin d'appliquer des heuristiques de recherche HTN à ces derniers. Nous avons comparé notre approche avec une approche basée sur les *timelines* partageant la même expressivité en termes des catégories de Cushing. Nous avons montré que TEP surpasse en termes de temps d'exécution pour trouver une solution, et que les deux approches sont comparables en ce qui concerne le makespan. Une manière d'améliorer l'approche TEP serait d'implémenter des heuristiques tenant compte du makespan dans TEP, ce qui permettrait d'améliorer la qualité des solutions retournées.

6 TTC : Temporal Task Converter

Dans le chapitre précédent, nous avons présenté TEP, un planificateur hiérarchique et temporel capable de résoudre des problèmes dans toutes les catégories de Cushing (voir Chapitre 5). Le principe de TEP est de relaxer un problème temporel de la durée de ses actions pour représenter le réseau de tâches temporelles comme un plan partiel de la planification POCL. L'intérêt est de pouvoir réutiliser les heuristiques de ce type de planification pour la recherche de plans solutions. Cependant, TEP reste lié au formalisme POCL et peut uniquement utiliser des processus et heuristiques propres à ce type de planification. L'objectif de **TTC** est de proposer une adaptation de TEP, c'est-à-dire d'encoder un problème hiérarchique et temporel en un problème non temporel qui pourra par la suite être résolu par n'importe quel planificateur hiérarchique non temporel. Pour résumer, TTC encode un problème hiérarchique et temporel en un problème hiérarchique non temporel.

L'encodage présenté ici est une généralisation de l'approche de TEP permettant d'utiliser un planificateur HDDL [Höl+20a] *générique* pour résoudre un problème hiérarchique temporel. Il repose donc sur le même principe de résolution que TEP, où les contraintes de durée du problème sont d'abord relaxées. La différence est que le problème relaxé est encodé comme un problème hiérarchique non temporel et non plus comme un plan partiel POCL. Ce problème encodé peut ensuite être résolu par n'importe quel planificateur HDDL. Dans un dernier temps, les contraintes de durée sont réinjectées dans la solution. Si aucun conflit n'est introduit par la réinjection de ces contraintes, la solution est renvoyée comme solution générale du problème. Enfin une étape de *déordonnancement* est possible afin d'augmenter la concurrence de la solution trouvée au prix du coût algorithmique de la procédure de déordonnancement. Un schéma du fonctionnement de TTC est représenté dans la Figure 6.1. La particularité de TTC est d'être capable de produire des plans temporels concurrents même en utilisant un planificateur hiérarchique produisant des plans séquentiels. Cette particularité reste vraie avec et sans l'étape de déordonnancement. Cette étape a pour but *d'augmenter* la concurrence dans la solution trouvée, mais n'est pas nécessaire à celle-ci. Il faut également noter que TTC se limite à la résolution

de problèmes dans la première et la deuxième catégorie de Cushing, et ne permet pas la résolution de ceux dans la troisième catégorie.

La suite de ce chapitre est organisée de la manière suivante, nous commençons par donner une définition du problème à résoudre, puis nous présentons concrètement l'encodage réalisé par TTC et enfin, nous terminons en présentant les résultats des expérimentations menées avec TTC.

6.1 Définition du problème

Tout comme TEP, l'ambition de TTC est de résoudre des problèmes tels que définis dans HDDL 2.1 [Pel+23]. La définition du problème est donc similaire à celle du chapitre précédent (voir Chapitre 5). Il faut noter que TTC se limite à un sous-ensemble des problèmes descriptibles par HDDL 2.1, comme nous le verrons par la suite.

Ainsi un **problème hiérarchique temporel** $P_t = (L, \mathcal{T}, A, M, I, tn)$ est composé d'un ensemble fini de propositions logiques L, d'un ensemble fini $\mathcal{T}$ de tâches temporelles, d'un ensemble fini d'actions temporelles A, d'un ensemble fini de méthodes M, et enfin d'un état initial I couplé à un réseau initial de tâches temporelles tn.

Un réseau de tâches temporelles $tn = (T, \prec, \alpha)$ est composé d'un ensemble fini T de symboles de tâches à résoudre, d'une fonction $\alpha : T \mapsto \mathcal{T}$ associant à chacun de ces symboles la tâche de $\mathcal{T}$, ainsi que d'un ensemble $\prec$ de contraintes de Allen [All83] portant sur les débuts et les fins des tâches de T. Comme dans le chapitre précédent, $\forall t \in T$, $v_t^s \in \mathbb{R}$ représente la variable temporelle associée au début de la tâche t et $v_t^e \in \mathbb{R}$ la variable temporelle associée à la fin de t.

Une **action temporelle** $a = (name(a), start(a), end(a), inv(a), d) \in A$ est composée de son nom $name(a)$, de deux actions instantanées $start(a)$ et $end(a)$ représentant respectivement le début et la fin de a, d'un ensemble de propositions $inv(a) \subset \mathcal{L}$ représentant les invariants de a, ainsi que de sa durée $d \in \mathbb{R}$.

Une **méthode temporelle** $m = (task(m), pre(m), tn(m)) \in M$ est composée du nom de la tâche qu'elle décompose $task(m)$, d'un ensemble de préconditions $pre(m) \in L$ ainsi que d'un réseau de tâches temporelles $tn(m)$ représentant le réseau résultant de la décomposition de $task(m)$ par m.

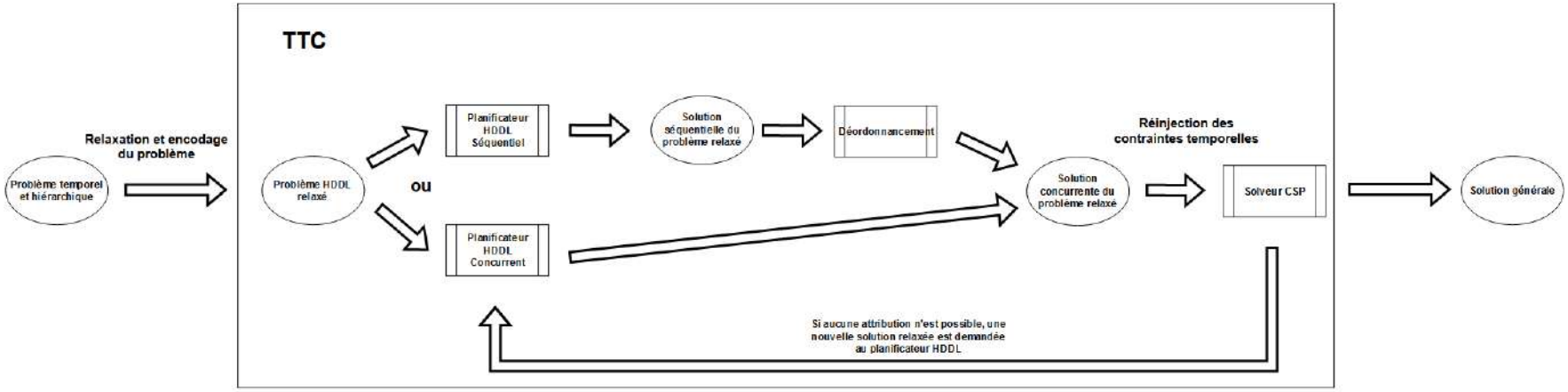

Figure 6.1. : Principe général de TTC, le problème hiérarchique temporel initial est encodé en un problème HDDL via le principe de relaxation des durées. Le problème encodé est résolu à l'aide d'une planificateur HDDL, les contraintes de durée sont réinjectées après une étape optionnelle de déordonnancement.

Si $tn = (T, \prec, \alpha)$ est un réseau de tâches temporelles et que $task(m) \in T$ alors le réseau de tâches temporelles $tn' = (T', \prec', \alpha')$ résultant de la décomposition de tn par m est défini par l'équation 5.1.

Enfin, un **plan temporel** $\pi = \{(a_1, t_1), (a_2, t_2), \ldots, (a_n, t_n)\}$ est un ensemble de couples d'actions temporelles et de dates. Un plan temporel est solution d'un problème temporel $P_t = (L, \mathcal{T}, A, M, I, tn)$ si :

1. $\forall i \in [\![1, n]\!]$, a_i est applicable sur $[t_i, t_i + d_{a_i}]$.
2. ce plan résout un réseau de tâches $tn_f = (T_f, \prec_f, \alpha_f)$ issu de la décomposition de tn par des méthodes de M.

Des détails supplémentaires pour la définition d'un plan solution sont disponibles à la définition 13.

6.2 Temporal Task Converter : une compilation non temporelle

Le principe général de **TTC** est similaire à celui présenté au chapitre précédent pour TEP. Les contraintes de durée des actions temporelles sont dans un premier temps relaxées. Le problème relaxé est ensuite résolu comme un problème non temporel et enfin les contraintes de durée sont réinjectées dans la solution potentielle. Cependant, à la différence de TEP, TTC réalise une compilation *concrète* du problème relaxé en un problème hiérarchique non temporel. Ce problème relaxé est concrètement réécrit comme un problème du formalisme HDDL [Höl+20a].

L'avantage de TTC par rapport à TEP est donc de pouvoir résoudre le problème relaxé au moyen d'un planificateur HDDL générique, et de ne plus dépendre du formalisme de résolution POCL. Nous verrons par la suite que cet aspect donne un avantage d'efficacité à TTC par rapport à TEP. En contrepartie, en réécrivant un problème hiérarchique temporel en un problème hiérarchique non temporel, TTC perd une partie de l'expressivité du formalisme de la planification temporelle. Ce faisant, une partie des problèmes temporels ne pourront pas être résolus par TTC. Nous verrons par la suite que les problèmes de la troisième catégorie de Cushing ne pourront pas être résolus par TTC.

L'objectif de cette section est d'expliciter la compilation du problème hiérarchique temporel relaxé en un problème hiérarchique non temporel. Pour cela, nous commençons par la compilation des actions duratives, puis nous explicitons la compilation des tâches abstraites et des réseaux de tâches temporelles.

Dans le reste de cette section, nous considérons un problème hiérarchique temporel $P_t = (L, \mathcal{T}, A, M, I, tn)$ et nous explicitons l'encodage de ce problème en un problème hiérarchique non temporel $P_h = (L_h, \mathcal{T}_h, A_h, M_h, I_h, tn_h)$.

6.2.1 Compilation des tâches abstraites

La compilation des tâches abstraites dans TTC repose sur la même idée que TEP. En effet, dans TTC, les tâches abstraites sont représentées par deux tâches non temporelles ordonnées, l'une abstraite et l'autre primitive. Ces deux tâches représentent respectivement le début et la fin de la tâche abstraite temporelle. La tâche abstraite non temporelle se décompose au moyen des mêmes méthodes que la tâche abstraite temporelle. La tâche primitive est une tâche sans préconditions ni effets et sert uniquement à supporter les contraintes de la fin de la tâche abstraite temporelle.

Plus formellement, chaque tâche abstraite $t \in \mathcal{T}$ est compilée en une tâche abstraite non temporelle $start_t$, en un tâche primitive end_t et en une action $noop_t = (end_t, \emptyset, \emptyset, \emptyset)$ et on a :

$$\begin{aligned} \{start_t, end_t \mid t \text{ une tâche abstraite de } \mathcal{T}\} &\subset \mathcal{T}_h \\ \{noop_t \mid t \text{ une tâche abstraite de } \mathcal{T}\} &\subset A_h \end{aligned} \tag{6.1}$$

Une représentation de cette compilation est illustrée dans la Figure 6.2. Dans ce schéma, la tâche t est compilée en une tâche abstraite s_t et une tâche primitive équivalente à l'action $noop_t$. Ces deux tâches instantanées sont ordonnées par une contrainte d'ordre de telle sorte que $s_t < noop_t$.

6.2.2 Compilation des actions duratives

La compilation des actions duratives dans TTC est similaire à celle de TEP présentée au chapitre précédent. En effet, une action durative est compilée comme deux actions instantanées a_s et a_e représentant respectivement le début et la fin de l'action durative. Les préconditions et les effets des actions instantanées encodées correspondent respectivement aux préconditions et aux effets de début et de fin

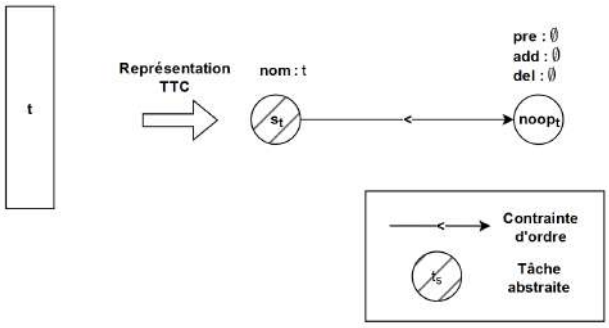

Figure 6.2. : Compilation d'une tâche abstraite par TTC : la tâche abstraite t est compilée en deux tâches, l'une abstraite et l'autre primitive.

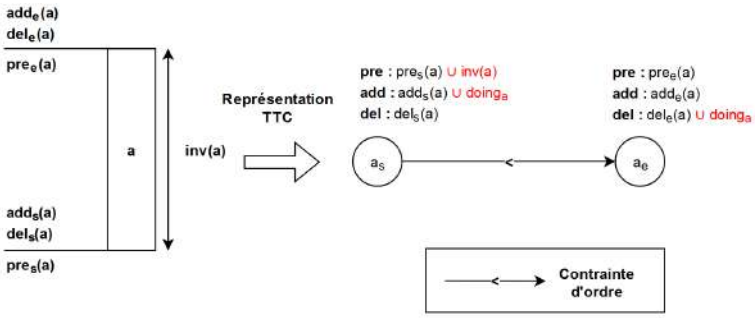

Figure 6.3. : Compilation d'une tâche primitive par TTC : l'action durative a est compilée en deux actions instantanées, a_s et a_e.

de l'action durative. De plus, les invariants de l'action durative sont ajoutés aux préconditions de l'action de début. Une représentation de cette compilation est disponible dans la Figure 6.3.

Notons que considérer les invariants d'une action durative comme des préconditions est *suffisant* pour produire des plans valides. Cependant, ça n'est pas une condition nécessaire, et ce faisant, TTC perd l'espoir de résoudre les problèmes de la troisième catégorie de Cushing.

Quand TEP préserve les invariants de l'action durative au moyen d'un lien causal, TTC ne peut pas faire de même (les liens causaux sont propres au formalisme POCL). Dans le but de contourner ce problème, TTC définit de nouveaux prédicats, un prédicat par action durative du problème initial. Chaque prédicat indique si l'action à laquelle il est associé est en cours d'exécution. Ce prédicat est ajouté aux effets positifs de l'action instantanée représentant le début de l'action durative et est également ajouté aux effets négatifs de l'action instantanée de fin. Par la suite, pour une action durative $a \in A$, on notera $doing_a$ ce prédicat.

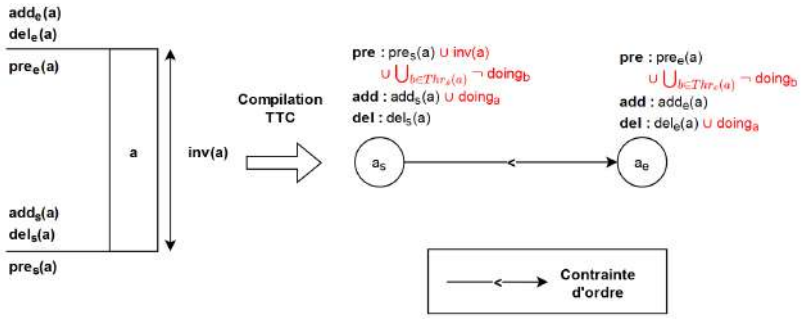

Figure 6.4. : Compilation complète d'une action durative par TTC : des préconditions ont été ajoutées afin de garantir la préservation des invariants.

Le rôle de ces prédicats est de contrôler que l'ajout d'une nouvelle action dans le plan n'invalide pas les invariants des actions déjà en cours d'exécution. Pour parvenir à cela, il nous faut d'abord définir, pour chaque action durative $a \in A$ du problème initial, deux ensembles d'actions duratives $Thr_s(a) \subset A$ et $Thr_e(a) \subset A$ contenant respectivement l'ensemble des actions duratives dont les invariants seront invalidés par les effets de début de a (dans le cas de $Thr_s(a)$) et les effets de fin de a (dans le cas de $Thr_e(a)$). Plus formellement, on peut définir ces deux ensembles de la manière suivante :

$$\begin{aligned} \forall a \in A, Thr_s(a) =& \{b \in A \mid inv(b) \cap del_s(a) \neq \emptyset\} \\ Thr_e(a) =& \{b \in A \mid inv(b) \cap del_e(a) \neq \emptyset\} \end{aligned} \tag{6.2}$$

Une fois ces ensembles déterminés, TTC ajoute de nouvelles préconditions aux actions instantanées encodées afin de garantir qu'une action ne peut se commencer ou se terminer uniquement si elle n'invalide pas les invariants d'une action en cours d'exécution. Pour une action durative $a \in A$, pour chaque action $b \in Thr_s(a)$, la précondition $\neg(doing_b)$ est ajoutée à l'action instantanée compilée a_s. De la même manière, la précondition $\neg(doing_b)$ est ajoutée à a_e pour chaque action durative de $Thr_e(a)$. Un schéma représentant la compilation complète d'une action durative est représentée dans la Figure 6.4

Plus formellement, pour chaque action durative $a \in A$, on encode deux actions instantanées :

$$s_a = (start_a, pre_s(a) \cup inv(a) \cup \bigcup_{b \in Thr_s(a)} \neg doing_b, add_s(a) \cup doing_a, del_s(a))$$

$$e_a = (end_a, pre_e(a) \cup \bigcup_{b \in Thr_e(a)} \neg doing_b, add_e(a), del_e(a) \cup doing_a)$$

Nous pouvons à présent définir entièrement les ensembles $\mathcal{T}_h$, A_h et L_h :

$$\begin{aligned}
\mathcal{T}_h =& \{start_t, end_t \mid t \text{ une tâche abstraite de } \mathcal{T}\} \\
& \cup \{start_a, end_a \mid a \in A\} \\
A_h =& \{noop_t \mid t \text{ une tâche abstraite de } \mathcal{T}\} \\
& \cup \{s_a, e_a \mid a \in A\} \\
L_h =& L \cup \{doing_a \mid a \in A\}
\end{aligned}$$

6.2.3 Compilation des réseaux de tâches temporelles

Nous allons à présent expliciter la compilation des réseaux de tâches temporelles dans TTC. Cette compilation est utilisée pour deux éléments du problème compilé : la définition du réseau de tâches initial tn_h ainsi que la définition des méthodes compilées de M_h.

La compilation des réseaux de tâches est également similaire à celle présentée au chapitre précédent pour TEP. Les tâches temporelles du réseau sont encodées par deux tâches telles que définies dans les deux sections précédentes (voir sections 6.2.1 et 6.2.2). Les contraintes d'ordres portant sur les tâches temporelles sont reportées sur les tâches non temporelles du problème encodé. Il faut cependant remarquer que le formalisme HDDL ne permet pas la définition de contraintes $\leq$. Ces contraintes sont donc remplacées par des contraintes strictes $<$ dans le problème non temporel.

Plus formellement, si $tn = (T, \prec, \alpha)$ est un réseau de tâches temporelles, TTC encode un réseau non temporel $ttc(tn) = (T_h, \prec_h, \alpha_h)$ avec :

$$\begin{aligned}
T_h =& \{start_t, end_t \mid t \in T\} \\
\prec_h =& \{(start_t < end_t) \mid t \in T\} \\
& \cup \{(start_{t1} < start_{t2}) \mid (v^s_{t1} \leq v^s_{t2}) \in \prec\} \\
& \cup \{(start_{t1} < end_{t2}) \mid (v^s_{t1} \leq v^e_{t2}) \in \prec\} \\
& \cup \{(end_{t1} < end_{t2}) \mid (v^e_{t1} \leq v^e_{t2}) \in \prec\} \\
& \cup \{(end_{t1} < start_{t2}) \mid (v^e_{t1} \leq v^s_{t2}) \in \prec\} \\
\alpha_h =& \{(start_t, start_{\alpha(t)}) \mid t \in T\} \\
& \cup \{(end_t, end_{\alpha(t)}) \mid t \in T\}
\end{aligned}$$

Finalement, nous pouvons définir l'ensemble M_h des méthodes encodées ainsi que tn_h le réseau initial du problème encodé :

$$M_h = \{m_h = (start_{task(m)}, pre(m), ttc(tn(m))) \mid m \in M\}$$
$$tn_h = ttc(tn)$$

6.3 Expérimentations

Dans cette section, nous présentons les résultats des expérimentations que nous avons menées avec TTC. L'objectif est de comparer les performances de TTC avec celles de TEP. Nous comparons également TTC au planificateur FAPE qui a également servi de référence de planificateur hiérarchique et temporel.

6.3.1 Configuration expérimentale

Tous les tests présentés par la suite ont été réalisés sur un seul cœur Intel Core i7-9850H d'une même machine. La limite de mémoire a été fixée à 8Go pour un temps de résolution limité à 600 secondes.

Nous utilisons les mêmes critères d'évaluation que pour le chapitre précédent, à savoir :

1. le *temps de résolution*, représentant le temps utilisé par la configuration pour produire un plan solution. Ce critère permet de mesurer l'efficacité de la configuration.
2. le *makespan* du plan solution, représentant la durée totale du plan. Ce critère permet une évaluation de la qualité du plan solution renvoyé.
3. la *couverture* de la configuration, représentant la proportion de problèmes résolus parmi l'ensemble des problèmes benchmarks.

Une nouvelle fois, le score IPC a été utilisé afin d'évaluer chacun des critères d'évaluation. Le détail du calcul de ce score est décrit dans la partie 5.3.1.

Pour le fonctionnement de TTC, ce dernier doit être couplé avec un planificateur HDDL. Nous avons sélectionné le planificateur PANDA [Höl+20b] pour ce rôle. En effet, ce planificateur s'est montré particulièrement polyvalent au cours des différentes compétitions internationales IPC [Val+15 ; BHB21], et ce pour des problèmes totalement ou partiellement ordonnés. Ce planificateur produisant des

plans *séquentiels*, une étape de déordonnencement est réalisée après la génération d'un plan solution séquentiel par PANDA. Cette étape a pour but d'éliminer certaines contraintes d'ordre d'un plan solution pour le transformer en plan partiellement ordonné. Dans le cas de TTC, réaliser cette étape a plusieurs avantages. Le premier est d'augmenter la qualité du plan solution en augmentant sa concurrence. Le second est de faciliter la réinjection des contraintes de durée une fois la solution trouvée. En effet, moins un plan solution est contraint et moins l'ajout de nouvelles contraintes ne risque d'invalider la solution.

Nous avons donc choisi d'utiliser la librairie MRR (Minimum Reinsantiated Reorder) [MBM16] qui est, à ce jour, le système de déordonnencement le plus efficace à notre connaissance. Il faut cependant noter que MRR déordonne un plan sans tenir compte des contraintes hiérarchiques issues de la décomposition des tâches. Ces contraintes sont maintenues dans le plan solution au travers d'un module approprié. Plus précisément, les contraintes d'ordres hiérarchiques sont sauvegardées par le module avant l'étape de déordonnencement, puis sont ajoutées à celles du plan partiellement ordonné issu de MRR.

Enfin pour toutes les configurations de TEP et pour TTC, le solveur CSP "Glop" de la librairie Or-Tools [PDG23] a été utilisée pour l'affectation temporelle des actions du plan solution.

6.3.2 Benchmarks de planification

Nous réutilisons ici les benchmarks de planification utilisés dans le chapitre précédent. Nous retrouvons les domaines du *Gripper*, du *Satellite* ainsi que du *Rover* qui sont des domaines de la seconde catégorie de Cushing. De plus, deux domaines de la troisième catégorie de Cushing avaient été utilisés lors du chapitre précédent : *AreaScan* et *TableCarriers*. Ces problèmes ne sont pas résolubles par TTC, cependant une version simplifiée de *AreaScan* (sans concurrence nécessaire) a été utilisée afin d'étendre l'ensemble des problèmes tests.

6.3.3 Résultats

Les résultats des expérimentations sont affichés sur le tableau 6.1. Comme cela est visible sur ce tableau, globalement, TTC se place derrière les différentes configurations de TEP mais reste supérieur au planificateur FAPE. Ce résultat est vrai à la fois pour le critère du temps de résolution que pour celui du makespan de la solution.

	Gripper			Satellite			Rover			AreaScan PO			AreaScan TO			Total		
	S. Time	Span	Cov.	S. Time	Span	Cov.	S. Time	Span	Cov.	S. Time	Span	Cov.	S. Time	Span	Cov.	S. Time	Span	Cov.
TEP(TC)	3,45	**7,94**	**7/10**	5,64	6,42	**8/9**	4,08	5,23	**10/10**	19,99	**20,81**	**22/22**	19,93	**20,76**	**22/22**	53,09	61,16	**68/73**
TEP(MME)	**3,49**	**7,94**	**7/10**	6,37	6,64	**8/9**	6,62	6,99	9/10	17,64	20,61	**22/22**	20,25	20,53	**22/22**	54,38	62,70	**68/73**
TEP(TDGm)	2,82	7,00	6/10	4,86	7,30	**8/9**	**8,56**	7,53	9/10	18,67	20,61	**22/22**	**21,71**	20,53	**22/22**	56,61	**62,98**	67/73
TEP(TC + #Flaw)	3,31	7,00	6/10	**6,71**	**7,53**	**8/9**	6,21	7,01	**10/10**	**20,39**	20,61	**22/22**	21,48	20,56	**22/22**	**58,09**	62,71	**68/73**
FAPE	1,13	7,00	6/10	2,74	5,78	6/9	3,64	**7,58**	8/10	10,00	17,93	19/22	13,69	20,51	**22/22**	31,19	58,81	61/73
TTC + Panda + MRR	3,45	**7,94**	**7/10**	4,83	5,91	7/9	4,08	5,23	**10/10**	19,99	**20,81**	**22/22**	19,93	**20,76**	**22/22**	52,28	60,65	**68/73**

Tableau 6.1. : Résultats des expérimentations pour les différentes configurations.

Pour ce qui est de la couverture cependant, TTC se place au même niveau que les meilleures configurations de TEP.

L'explication de ces résultats pourrait reposer sur le fait que TTC dépend de plusieurs systèmes et librairies pour fonctionner, ce qui implique des communications entre ceux-ci. La transmission des résultats d'un système à l'autre peut être coûteuse d'un point de vue du temps de résolution. Ce problème est exacerbé lorsqu'aucune affectation temporelle n'est possible par le solveur CSP et que la boucle de rétroaction est enclenchée (voir Figure 6.1 pour un schéma du fonctionnement de TTC).

Il faut cependant rappeler que TTC fonctionne avec un planificateur HDDL générique. Il permet donc à un planificateur non temporel de rivaliser avec des planificateurs spécifiquement temporels tels que FAPE à moindre coût. De plus, les améliorations dans les domaines de la planification hiérarchique non temporelle et du déordonnencement sont aussi des améliorations directes apportées à TTC.

6.4 Conclusion

Dans ce chapitre, nous avons présenté TTC, une méthode de compilation permettant de résoudre des problèmes de planification hiérarchique et temporelle en utilisant des planificateurs hiérarchiques non temporels. Nous avons pu constater que TTC obtient des résultats comparables à ceux de systèmes temporels tout en utilisant des systèmes non temporels. TTC est donc une méthode générique de planification temporelle permettant d'éviter le coût de développement d'un système spécifiquement adapté.

Plusieurs pistes restent cependant à explorer. Tout d'abord, de nouveaux tests avec d'autres planificateurs hiérarchiques seraient à mener afin de déterminer lesquels sont les plus adaptés à TTC. De plus, l'ajout de conditions et contraintes supplémentaires dans le problème encodé par TTC pourrait permettre de mieux contrôler la faisabilité de la solution relaxée, et ainsi permettre de réduire le nombre d'appels à la boucle de rétroaction.

7 Conclusions et perspectives

L'objet de ce chapitre est de résumer les diverses contributions présentées au cours des chapitres précédents, d'en présenter les limites et enfin d'envisager les perspectives d'évolution possibles pour ces travaux.

7.1 Bilan des contributions

Nous avons présenté trois contributions. Ces contributions sont à l'image de la démarche incrémentales qui a été suivie au cours de cette thèse. Dans le Chapitre 4, nous avons présenté **CTHD** une méthode d'encodage d'un problème de planification hiérarchique en un problème de planification classique. La particularité de l'encodage étant qu'une solution *séquentielle* du problème encodé décrit une solution *concurrente* du problème hiérarchique initial. Ainsi, CTHD est une méthode générale permettant à des planificateurs séquentiels classiques de générer des solutions concurrentes pour des problèmes hiérarchiques. Nous avons montré expérimentalement que cette spécificité permet à CTHD de produire des plans de meilleure qualité tout en restant compétitive avec les encodages de sa catégorie. CTHD a permis d'appréhender les méthodes de planification permettant de produire un plan solution concurrent, d'abord dans un contexte de planification non temporelle.

L'ajout de la composante temporelle est l'objet du Chapitre 5. Nous avons présenté **TEP**, un système de planification temporelle de troisième catégorie. La spécificité de ce système est d'adapter une technique de planification non temporelle à la planification temporelle : la planification POCL. Cet aspect permet à TEP d'utiliser les algorithmes et heuristiques propres à la planification POCL dans le cadre temporel, sans pour autant réduire l'expressivité de la planification temporelle. Nos expérimentations ont montré que TEP est plus performant que les autres systèmes de planification partageant son expressivité. L'ambition de TEP est de se rapprocher autant que possible d'un système de planification non temporel afin de bénéficier de l'efficacité de leurs heuristiques de recherche. Cependant, une conversion directe n'est cependant pas possible du fait de problèmes temporels non exprimables dans le formalisme non temporel, les problèmes de la troisième catégorie de Cushing.

Le Chapitre 6 étudie les problèmes n'appartenant pas à cette catégorie. Nous y présentons **TTC**, une méthode générale de résolution de problèmes hiérarchiques et temporels en utilisant des systèmes de planification hiérarchique et non temporelle. Bien qu'une partie de l'expressivité de la planification temporelle ne soit pas modélisée par TTC, cette méthode permet à tout planificateur hiérarchique non temporel de résoudre des problèmes hiérarchiques temporels également. Nos expérimentations ont montré que TTC permet d'obtenir des résultats comparables à TEP, autant en termes de qualité de plans que de temps d'exécution, le tout en utilisant un système de planification non temporelle.

Le point commun de ces contributions est d'essayer de traduire, du mieux possible, un problème de planification en un nouveau problème de complexité inférieure. Ce type de traduction présente plusieurs avantages. Tout d'abord, d'un point de vue algorithmique, cela permet d'utiliser les systèmes et/ou techniques de résolutions propres au nouveau problème. Dans le cas de la planification automatique, cela signifie souvent bénéficier de travaux et recherches plus nombreux. D'un point de vue scientifique, cela permet d'exposer les différences entre les formalismes de planification :

— Le cas de **CTHD** expose une différence *structurelle* des problèmes de planification classique et hiérarchique. En effet, si l'espace de recherche associé à un problème de planification classique est toujours *fini*, l'espace de recherche associé à un problème hiérarchique est possiblement infini [EHN96]. Ce résultat se retrouve dans CTHD au travers de la *progression bound*, qui limite la taille du réseau de tâches et du même coup réduit l'espace de recherche à un ensemble fini.
— Les différences entre la planification temporelle et non temporelle sont mises en évidence par **TEP** et **TTC**. En effet, en relaxant les contraintes de durée et en introduisant le concept de *postcondition*, TEP est capable de représenter un réseau de tâches temporelles comme un plan partiel de la planification POCL (non temporelle). La différence entre les formalismes temporels et non temporels repose donc entre ces deux notions. A fortiori, TTC a montré qu'en (1) relaxant les contraintes de durée du problème et en (2) considérant les *postconditions* comme des *préconditions*, alors un problème temporel devient directement exprimable en un problème non temporel.

La deuxième contribution apportée par les travaux présentés ici porte sur le *coût de la concurrence*. En effet, relativement à la quantité de travaux effectués dans le domaine de la planification automatique, un faible nombre traite ou aborde le sujet de la concurrence. Les contributions réalisées ici permettent d'apporter une appréciation

du *coût* à payer pour pouvoir produire des plans concurrents plutôt que séquentiels. Tout d'abord, CTHD augmente une approche séquentielle, HTN2STRIPS [Alf+16], afin de permettre la construction de plans concurrents. La comparaison entre ces deux approches permet de montrer que si le coût de la concurrence est effectivement non nul, il ne signifie pas une perte d'efficacité drastique pour le planificateur. La comparaison entre TEP et TTC a, quant à elle, permis de comparer deux approches de la génération de plans concurrents : produire *directement* un plan concurrent ou désordonner un plan séquentiel. Nos expérimentations ont montré que si les deux approches sont comparables en termes d'efficacité, la gestion interne de la concurrence permet de mieux adapter les algorithmes et heuristiques de recherche à la création de tels plans.

7.2 Limites et perspectives d'évolution

Pour terminer, nous présentons ici les différentes limites des travaux présentés dans ce manuscrit, et, nous proposons des perspectives d'amélioration pouvant être envisagées. Ces limites sont de natures différentes et peuvent être présentées de la manière suivante.

Limites liées à la reformulation des problèmes : Nos travaux ont portés sur la réécriture de problèmes de planification en problèmes d'expressivité plus faible. Si cette reformulation permet d'utiliser les méthodes propres à la nouvelle catégorie, elle ne permet pas d'améliorer les techniques de résolutions de la catégorie supérieure. Avec les méthodes par reformulation, il est également difficile d'orienter la recherche de solution vers un plan *optimal* quand le critère d'optimalité est propre à la catégorie supérieure. Par exemple, les heuristiques utilisées dans TEP se concentrent sur la résolution du problème causal et ne prennent pas en compte la dimension temporelle du problème. Dans ces conditions, TEP ne peut pas garantir que la solution générée sera optimale en termes de durée du plan. Une des perspectives d'évolution serait donc d'étudier des techniques de planification ou des heuristiques permettant à la fois de résoudre le problème causal tout en tenant compte de la dimension temporelle du problème.

Limites liées au formalisme de planification : Une des limites majeures de CTHD, TEP ou TTC est la non gestion des propriétés numériques. En d'autres termes, nos travaux supposent les propriétés de l'environnement à valeurs booléennes, or de

nombreux problèmes concrets nécessitent des propriétés à valeurs dans des espaces dénombrables (finis ou infinis) voire indénombrables. Si certains planificateurs sont capables de prendre en compte ce type de propriétés, ce n'est pas le cas de la majorité ni même des approches présentées ici. La seconde limite est liée au langage de planification même, qui pourrait recevoir des améliorations. Par exemple, on pourrait imaginer des méthodes permettant de résoudre plusieurs tâches, ou même tout un réseau de tâches d'un seul coup.

Limites de la planification centralisée : Nous sommes ici placés dans un cadre de planification centralisée, où l'on suppose une connaissance totale et exacte de l'environnement. Ceci est une limitation dans les cadres d'application concrets où un module de *replanification*, ou de réparation de plan, est également requis pour assurer le bon déroulement de l'opération. Cela passe par des systèmes permettant la communication d'informations et/ou la prise individuelle de décisions. En d'autres termes, les systèmes proposés ici se concentrent sur la planification globale plutôt que sur la coopération d'appareils. Des travaux autour notamment de la séparation du problème en sous-problèmes, couplée à une résolution décentralisée et à une méthode de fusion des différents plans pourrait être envisagée afin de construire un plan solution de manière collaborative.

Limites de la planification indépendante du domaine : Tous les systèmes et heuristiques proposés ici sont indépendants du domaine, c'est à dire que leur fonctionnement est le même quel que soit le domaine de planification rencontré. Cette particularité offre une grande flexibilité des approches. Cependant, une piste d'amélioration serait de garder des algorithmes de résolution indépendants mais de les coupler avec des heuristiques de recherche qui seraient quant à elles *spécialisées* dans un domaine de planification particulier. On peut par exemple imaginer utiliser des techniques d'apprentissage pour améliorer les performances du système sur ce domaine particulier.

Bibliographie

[Alf+16] Ron ALFORD, Gregor BEHNKE, Daniel HÖLLER, Pascal BERCHER, Susanne BIUNDO et David AHA. “Bound to plan : Exploiting classical heuristics via automatic translations of tail-recursive HTN problems”. In : *International Conference on Automated Planning and Scheduling (ICAPS)*. T. 26. 2016, p. 20-28 (cf. p. 66, 76, 84, 91, 93, 135).

[AKN09] Ronald ALFORD, Ugur KUTER et Dana NAU. “Translating HTNs to PDDL : A Small Amount of Domain Knowledge Can Go a Long Way.” In : *International Joint Conference on Artificial Intelligence (IJCAI)*. T. 9. 2009, p. 1629-1634 (cf. p. 44, 66, 75).

[All83] James ALLEN. “Maintaining knowledge about temporal intervals”. In : *Communications of the ACM* 26.11 (1983), p. 832-843 (cf. p. 52, 98, 122).

[AHL07] Maud AMATE, Alain HÉTET et Michel LEERIS. “Le Sonar à Antenne Synthétique (SAS), application à la guerre des mines”. In : *Acoustique & techniques* 48 (2007), p. 23-28 (cf. p. 14).

[Arc17] Deana ARCHAMBAULT. “A Roadmap of the Future of Mine Countermeasures”. Thèse de doct. Monterey, California : Naval Postgraduate School, 2017 (cf. p. 3).

[Asu+05] Marc de la ASUNCIÓN, Luis CASTILLO, Juan FDEZ-OLIVARES, Óscar GARCIA-PÉREZ, Antonio GONZÁLEZ et Francisco PALAO. “SIADEX : An interactive knowledge-based planner for decision support in forest fire fighting”. In : *AI Communications* 18.4 (2005), p. 257-268 (cf. p. 68).

[BC04] Magali BARBIER et Elodie CHANTHERY. “Autonomous mission management for unmanned aerial vehicles”. In : *Aerospace science and technology* 8.4 (2004), p. 359-368 (cf. p. 34).

[Bar+17] Marco BARONI, Armand JOULIN, Allan JABRI, German KRUSZEWSKI, Angeliki LAZARIDOU, Klemen SIMONIC et Tomas MIKOLOV. “CommAI : Evaluating the first steps towards a useful general AI”. In : *ICLR Workshop Track*. 2017 (cf. p. 37).

[Bar+12] Javier BARREIRO, Matthew BOYCE, Minh DO, Jeremy FRANK, Michael IATAURO, Tatiana KICHKAYLO, Paul MORRIS, James ONG, Emilio REMOLINA, Tristan SMITH et David SMITH. “EUROPA : A platform for AI planning, scheduling, constraint programming, and optimization”. In : *4th International Competition on Knowledge Engineering for Planning and Scheduling (ICKEPS)* (2012) (cf. p. 69).

[Bec+14] Patrick BECHON, Magali BARBIER, Guillaume INFANTES, Charles LESIRE et Vincent VIDAL. “HiPOP : Hierarchical Partial-Order Planning.” In : *European Starting AI Researchers Symposium (STAIRS)*. 2014, p. 51-60 (cf. p. 67).

[Bed+05] Tania BEDRAX-WEISS, Conor MCGANN, Andrew BACHMANN, Will EDGINGTON et Michael IATAURO. *EUROPA2 : User and contributor guide*. Rapp. tech. NASA Ames Research Center, 2005 (cf. p. 40).

[BHB21] Gregor BEHNKE, Daniel HÖLLER et Pascal BERCHER, éd. *Proceedings of the 10th International Planning Competition : Planner and Domain Abstracts – Hierarchical Task Network (HTN) Planning Track (IPC 2020)*. 2021 (cf. p. 40, 63, 129).

[BHB18] Gregor BEHNKE, Daniel HÖLLER et Susanne BIUNDO. “totSAT-Totally-ordered hierarchical planning through SAT”. In : *AAAI Conference on Artificial Intelligence*. T. 32. 1. 2018 (cf. p. 63).

[Beh+22] Gregor BEHNKE, Florian POLLITT, Daniel HÖLLER, Pascal BERCHER et Ron ALFORD. “Making translations to classical planning competitive with other HTN planners”. In : *AAAI Conference on Artificial Intelligence*. T. 36. 9. 2022, p. 9687-9697 (cf. p. 66, 76, 86, 87, 90, 91).

[Ber+17] Pascal BERCHER, Gregor BEHNKE, Daniel HÖLLER et Susanne BIUNDO. “An admissible HTN planning heuristic”. In : *International Joint Conference on Artificial Intelligence (IJCAI)*. 2017, p. 480-488 (cf. p. 102, 112, 114, 115).

[BKB14] Pascal BERCHER, Shawn KEEN et Susanne BIUNDO. “Hybrid planning heuristics based on task decomposition graphs”. In : *International Symposium on Combinatorial Search (SoCS)*. 2014, p. 35-43 (cf. p. 102, 106, 112, 114, 115).

[Bia+09] Xinqian BIAN, Tao CHEN, Zheping YAN et Zheng QIN. “Autonomous mission management and intelligent decision for AUV”. In : *International Conference on Mechatronics and Automation (ICMA)*. IEEE. 2009, p. 2101-2106 (cf. p. 30).

[Bit+20] Arthur BIT-MONNOT, Malik GHALLAB, Félix INGRAND et David SMITH. “FAPE : a constraint-based planner for generative and hierarchical temporal planning”. In : *arXiv preprint arXiv :2010.13121* (2020) (cf. p. 69, 115).

[BF97] Avrim BLUM et Merrick FURST. “Fast planning through planning graph analysis”. In : *Artificial intelligence* 90.1-2 (1997), p. 281-300 (cf. p. 55).

[BST11] Dominik BÖHNLEIN, Katharina SCHWEIGER et Axel TUMA. “Multi-agent-based transport planning in the newspaper industry”. In : *International Journal of Production Economics* 131.1 (2011), p. 146-157 (cf. p. 30).

[BK04] Abdelbaki BOUGUERRA et Lars KARLSSON. “Hierarchical task planning under uncertainty”. In : *3rd Italian Workshop on Planning and Scheduling (IPS)*. Citeseer. 2004 (cf. p. 64).

[Car+20] Yaniel CARRENO, Èric PAIRET, Yvan PETILLOT et Ronald PETRICK. “Task allocation strategy for heterogeneous robot teams in offshore missions”. In : *International Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*. 2020, p. 222-230 (cf. p. 34).

[Cas+14] Michael CASHMORE, Maria FOX, Tom LARKWORTHY, Derek LONG et Daniele MAGAZZENI. “AUV mission control via temporal planning”. In : *International conference on robotics and automation (ICRA)*. IEEE. 2014, p. 6535-6541 (cf. p. 34).

[CPF23a] Nicolas CAVREL, Damien PELLIER et Humbert FIORINO. “Efficient HTN to STRIPS Encodings for Concurrent Planning”. In : *International Conference on Tools with Artificial Intelligence (ICTAI)*. IEEE. 2023, p. 962-969 (cf. p. 4, 73, 94).

[CPF23b] Nicolas CAVREL, Damien PELLIER et Humbert FIORINO. “On Guiding Search in HTN Temporal Planning with non Temporal Heuristics”. In : *ICAPS Workshop on Hierarchical Planning (HPlan)*. 2023, p. 28-34 (cf. p. 5, 73).

[Cha+08] Guillaume CHASLOT, Mark WINANDS, Jaap van den HERIK, Jos UITERWIJK et Bruno BOUZY. “Progressive strategies for Monte-Carlo tree search”. In : *New Mathematics and Natural Computation* 4.03 (2008), p. 343-357 (cf. p. 64).

[Col+10] Amanda COLES, Andrew COLES, Maria FOX et Derek LONG. “Forward-chaining partial-order planning”. In : *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*. T. 20. 2010, p. 42-49 (cf. p. 34).

[Col+09] Andrew COLES, Maria FOX, Keith HALSEY, Derek LONG et Amanda SMITH. “Managing concurrency in temporal planning using planner-scheduler interaction”. In : *Artificial Intelligence* 173.1 (2009), p. 1-44 (cf. p. 68).

[Coo23] Stephen COOK. “The complexity of theorem-proving procedures”. In : *Logic, Automata, and Computational Complexity : The Works of Stephen A. Cook*. 2023, p. 143-152 (cf. p. 26, 62).

[CT91] Ken CURRIE et Austin TATE. “O-plan : the open planning architecture”. In : *Artificial intelligence* 52.1 (1991), p. 49-86 (cf. p. 67).

[Cus+07] William CUSHING, Daniel WELD, Subbarao KAMBHAMPATI et Kartik TALAMADUPULA. “Evaluating temporal planning domains.” In : *International Conference on Automated Planning and Scheduling (ICAPS)*. 2007, p. 105-112 (cf. p. 54, 56, 57, 61).

[Dai+20] Wei DAI, Huimin LU, Junhao XIAO, Zhiwen ZENG et Zhiqiang ZHENG. “Multi-robot dynamic task allocation for exploration and destruction”. In : *Journal of Intelligent & Robotic Systems* 98 (2020), p. 455-479 (cf. p. 33).

[DR59] George DANTZIG et John Hubert RAMSER. “The Truck Dispatching Problem”. In : *Management Science* 6 (1959), p. 80-91 (cf. p. 28).

[DLA15] Lavindra DE SILVA, Raphaël LALLEMENT et Rachid ALAMI. “The HATP hierarchical planner : Formalisation and an initial study of its usability and practicality”. In : *International conference on intelligent robots and systems (IROS)*. IEEE. 2015, p. 6465-6472 (cf. p. 44, 62).

[DN10] Vladimir DJAPIC et Dula NAD. “Using Collaborative Autonomous Vehicles in Mine Countermeasures”. In : *OCEANS’10*. 2010, p. 1-7 (cf. p. 4).

[DG97] Marco DORIGO et Luca Maria GAMBARDELLA. “Ant colony system : a cooperative learning approach to the traveling salesman problem”. In : *IEEE Transactions on evolutionary computation* 1.1 (1997), p. 53-66 (cf. p. 28).

[Dvo+14] Filip DVORAK, Arthur BIT-MONNOT, Félix INGRAND et Malik GHALLAB. “A flexible ANML actor and planner in robotics”. In : *Planning and Robotics Workshop (PlanRob)*. 2014 (cf. p. 69).

[EN69] George ERNST et Allen NEWELL. *GPS : A case study in generality and problem solving*. Academic Press, Inc, 1969 (cf. p. 30).

[EHN96] Kutluhan EROL, James HENDLER et Dana NAU. “Complexity results for HTN planning”. In : *Annals of Mathematics and Artificial Intelligence* 18.1 (1996), p. 69-93 (cf. p. 65, 134).

[EHN94] Kutluhan EROL, James HENDLER et Dana NAU. “UMCP : A Sound and Complete Procedure for Hierarchical Task-network Planning.” In : *International Conference on Artificial Intelligence Planning Systems (AIPS)*. T. 94. 1994, p. 249-254 (cf. p. 63).

[FN71] Richard FIKES et Nils NILSSON. “STRIPS : A new approach to the application of theorem proving to problem solving”. In : *Artificial intelligence* 2.3-4 (1971), p. 189-208 (cf. p. 38).

[Flo+03] Franck FLORIN, François van ZEEBROECK, Isabelle QUIDU et Naig LE BOUFFANT. “Classification performances of Mine Hunting Sonar : Theory, practical results and operational applications”. In : *Undersea Defence Technology (UDT) Europe 2003*. 2003 (cf. p. 17).

[FL02] Maria FOX et Derek LONG. “PDDL+ : Modeling continuous time dependent effects”. In : *3rd International NASA Workshop on Planning and Scheduling for Space*. T. 4. 2002, p. 34-43 (cf. p. 40).

[FL03] Maria FOX et Derek LONG. “PDDL2.1 : An extension to PDDL for expressing temporal planning domains”. In : *Journal of Artificial Intelligence Research (JAIR)* 20 (2003), p. 61-124 (cf. p. 40, 51, 100).

[FJ03] Jeremy FRANK et Ari JÓNSSON. “Constraint-based attribute and interval planning”. In : *Constraints* 8 (2003), p. 339-364 (cf. p. 43).

[Fuk+97] Alex FUKUNAGA, Gregg RABIDEAU, Steve CHIEN et David YAN. “ASPEN : A framework for automated planning and scheduling of spacecraft control and operations”. In : *International Symposium on AI, Robotics and Automation in Space*. 1997, p. 181-187 (cf. p. 43).

[GLB12] Thibault GATEAU, Charles LESIRE et Magali BARBIER. “HiDDeN, une architecture décisionnelle distribuée pour la coopération de véhicules individuellement autonomes”. In : *Reconnaissance des Formes et Intelligence Artificielle (RFIA)*. 2012, p. 978-986 (cf. p. 67).

[Ge+21] Qiang GE, Fengxue RUAN, Baojun QIAO, Qian ZHANG, Xianyu ZUO et Lanxue DANG. “Side-Scan Sonar Image Classification Based on Style Transfer and Pre-Trained Convolutional Neural Networks”. In : *Electronics* 10.15 (2021), p. 1823-1829 (cf. p. 17).

[GB11] Thomas GEIER et Pascal BERCHER. “On the decidability of HTN planning with task insertion”. In : *International Joint Conference on Artificial Intelligence (IJCAI)*. T. 22. 3. 2011, p. 1955-1961 (cf. p. 41).

[Gen+13] L GENG, YF ZHANG, JJ WANG, Jerry FUH et SH TEO. “Mission planning of autonomous UAVs for urban surveillance with evolutionary algorithms”. In : *International Conference on Control and Automation (ICCA)*. IEEE. 2013, p. 828-833 (cf. p. 28).

[Ger+08] Alfonso GEREVINI, Ugur KUTER, Dana NAU, Alessandro SAETTI et Nathaniel WAISBROT. “Combining domain-independent planning and HTN planning : The duet planner”. In : *European Conference on Artificial Intelligence (ECAI)*. IOS Press, 2008, p. 573-577 (cf. p. 64).

[GL05] Alfonso GEREVINI et Derek LONG. *Plan constraints and preferences in PDDL3*. Rapp. tech. Department of Electronics for Automation, 2005 (cf. p. 40).

[Ger+09] Alfonso E GEREVINI, Patrik HASLUM, Derek LONG, Alessandro SAETTI et Yannis DIMOPOULOS. “Deterministic planning in the fifth international planning competition : PDDL3 and experimental evaluation of the planners”. In : *Artificial Intelligence* 173.5-6 (2009), p. 619-668 (cf. p. 40).

[GM04] Brian GERKEY et Maja MATARIĆ. “A formal analysis and taxonomy of task allocation in multi-robot systems”. In : *International Journal of Robotics Research (IJRR)* 23.9 (2004), p. 939-954 (cf. p. 33).

[GNT04] Malik GHALLAB, Dana NAU et Paolo TRAVERSO. *Automated Planning : theory and practice*. Elsevier, 2004 (cf. p. 26, 31, 37, 40, 79, 80).

[Gol+16] Robert GOLDMAN, Daniel BRYCE, Michael PELICAN, David MUSLINER et Kyungmin BAE. “A hybrid architecture for correct-by-construction hybrid planning and control”. In : *NASA Formal Methods : 8th International Symposium*. Springer. 2016, p. 388-394 (cf. p. 64).

[GK19] Robert GOLDMAN et Ugur KUTER. “Hierarchical task network planning in common Lisp : the case of SHOP3”. In : *12th European Lisp Symposium (ELS)*. Sous la dir. de Nicolas NEUSS. ELSAA, 2019, p. 73-80 (cf. p. 65).

[GMA18] Antoine GRÉA, Laetitia MATIGNON et Samir AKNINE. “HEART : HiErarchical Abstraction for Real-Time partial order causal link planning”. In : *ICAPS Workshop on Hierarchical Planning (HPlan)*. 2018, p. 17-25 (cf. p. 67).

[Hel06] Maltet HELMER. “The Fast Downward planning system”. In : *Journal of Artificial Intelligence Research (JAIR)* 26 (2006), p. 191-246 (cf. p. 91).

[HN01] Jörg HOFFMANN et Bernhard NEBEL. “The FF planning system : Fast plan generation through heuristic search”. In : *Journal of Artificial Intelligence Research (JAIR)* 14 (2001), p. 253-302 (cf. p. 91).

[HPS04] Jörg HOFFMANN, Julie PORTEOUS et Laura SEBASTIA. “Ordered landmarks in planning”. In : *Journal of Artificial Intelligence Research (JAIR)* 22 (2004), p. 215-278 (cf. p. 65).

[HKM10] Chad HOGG, Ugur KUTER et Héctor MUNOZ-AVILA. “Learning methods to generate good plans : Integrating HTN learning and reinforcement learning”. In : *AAAI Conference on Artificial Intelligence*. T. 24. 1. 2010, p. 1530-1535 (cf. p. 65).

[Höl+21] Daniel HÖLLER, Gregor BEHNKE, Pascal BERCHER et Susanne BIUNDO. “The PANDA framework for hierarchical planning”. In : *KI-Künstliche Intelligenz* (2021), p. 1-6 (cf. p. 67).

[Höl+20a] Daniel HÖLLER, Gregor BEHNKE, Pascal BERCHER, Susanne BIUNDO, Humbert FIORINO, Damien PELLIER et Ron ALFORD. “HDDL : An extension to PDDL for expressing hierarchical planning problems”. In : *AAAI conference on artificial intelligence*. T. 34. 06. 2020, p. 9883-9891 (cf. p. 44, 121, 124).

[Höl+20b] Daniel HÖLLER, Pascal BERCHER, Gregor BEHNKE et Susanne BIUNDO. “HTN planning as heuristic progression search”. In : *Journal of Artificial Intelligence Research (JAIR)* 67 (2020), p. 835-880 (cf. p. 129).

[How+98] Adele HOWE, Craig KNOBLOCK, Drew MCDERMOTT, Ashwin RAM, Manuela VELOSO, Daniel WELD, David WILKINS, Anthony BARRETT et Dave CHRISTIANSON. “PDDL | The Planning Domain Definition Language”. In : *Technical Report, Tech. Rep.* (1998) (cf. p. 39, 44).

[JJP15] Sergio JIMÉNEZ, Anders JONSSON et Héctor PALACIOS. “Temporal planning with required concurrency using classical planning”. In : *International Conference on Automated Planning and Scheduling (ICAPS)*. T. 25. 2015, p. 129-137 (cf. p. 68).

[JP94] David JOSLIN et Martha POLLACK. “Least-Cost Flaw Repair : A plan refinement strategy for partial-order Planning”. In : *AAAI Conference on Artificial Intelligence*. 1994, p. 1004-1009 (cf. p. 115).

[Kar+14] Divas KARIMANZIRA, Marco JACOBI, Torsten PFÜTZENREUTER, Thomas RAUSCHENBACH, Mike EICHHORN, Ralf TAUBERT et Christoph AMENT. “First testing of an AUV mission planning and guidance system for water quality monitoring and fish behavior observation in net cage fish farming”. In : *Information Processing in Agriculture* 1.2 (2014), p. 131-140 (cf. p. 30).

[KBM22] Maksim KENZIN, Igor BYCHKOV et Nikolai MAKSIMKIN. “A hierarchical approach to intelligent mission planning for heterogeneous fleets of Autonomous Underwater Vehicles”. In : *Journal of Marine Science and Engineering* 10.11 (2022), p. 1639-1656 (cf. p. 35).

[KS06] Levente KOCSIS et Csaba SZEPESVÁRI. “Bandit based monte-carlo planning”. In : *European Conference on Machine Learning (ECML)*. Springer. 2006, p. 282-293 (cf. p. 64).

[KN04] Ugur KUTER et Dana NAU. “Forward-chaining planning in nondeterministic domains”. In : *AAAI Conference on Artificial Intelligence*. 2004, p. 513-518 (cf. p. 64).

[Kut+05] Ugur KUTER, Dana NAU, Marco PISTORE et Paolo TRAVERSO. “A hierarchical task-network planner based on symbolic model checking.” In : *International Conference on Automated Planning and Scheduling (ICAPS)*. 2005, p. 300-309 (cf. p. 64).

[LSA18] Raphaël LALLEMENT, Lavindra de SILVA et Rachid ALAMI. “HATP : Hierarchical Agent-based Task Planner”. In : *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. 2018, p. 1823-1825 (cf. p. 62).

[LM14] Peter K LEHARDY et Christopher MOORE. “Deep Ocean Search for Malaysia Airlines flight 370”. In : *2014 Oceans-St. John's*. IEEE. 2014, p. 1-4 (cf. p. 3).

[Lem04] Solange LEMAI. “IxTeT-eXeC : planning, plan repair and execution control with time and resource management”. Thèse de doct. Institut National Polytechnique de Toulouse-INPT, 2004 (cf. p. 69).

[LA21] Charles LESIRE et Alexandre ALBORE. “PYHIPOP-Hierarchical Partial-Order Planner”. In : *Workshop on the International Planning Competition (WIPC)*. 2021 (cf. p. 67).

[LGW05] Feiyue LI, Bruce GOLDEN et Edward WASIL. “Very large-scale vehicle routing : new test problems, algorithms, and results”. In : *Computers & Operations Research* 32.5 (2005), p. 1165-1179 (cf. p. 29).

[Lin+20] Sun LIN, Zhu ANSHI, Li BO et Fan XIAOSHI. “HTN guided adversarial planning for RTS games”. In : *International Conference on Mechatronics and Automation (ICMA)*. IEEE. 2020, p. 1326-1331 (cf. p. 64).

[MMD20] Maurício MAGNAGUAGNO, Felipe Rech MENEGUZZI et Lavindra DE SILVA. “HyperTensioN : A three-stage compiler for planning”. In : *International Conference on Automated Planning and Scheduling (ICAPS)*. 2020 (cf. p. 65).

[Mah+18] Somaiyeh MAHMOUDZADEH, David POWERS, Karl SAMMUT, Adham ATYABI et Amirmehdi YAZDANI. “A hierarchal planning framework for AUV mission management in a spatiotemporal varying ocean”. In : *Computers & electrical engineering* 67 (2018), p. 741-760 (cf. p. 29).

[Mah+15] Somaiyeh MAHMOUDZADEH, David POWERS, Karl SAMMUT, Andrew LAMMAS et Amir Mehdi YAZDANI. “Optimal route planning with prioritized task scheduling for AUV missions”. In : *International Symposium on Robotics and Intelligent Sensors (IRIS)*. IEEE. 2015, p. 7-14 (cf. p. 29, 30).

[MPY16] Somaiyeh MAHMOUDZADEH, David POWERS et Amir Mehdi YAZDANI. “A novel efficient task-assign route planning method for AUV guidance in a dynamic cluttered environment”. In : *Congress on Evolutionary Computation (CEC)*. IEEE. 2016, p. 678-684 (cf. p. 29, 30).

[MRW08] Bhaskara MARTHI, Stuart RUSSELL et Jason Andrew WOLFE. “Angelic Hierarchical Planning : Optimal and Online Algorithms.” In : *International Conference on Automated Planning and Scheduling (ICAPS)*. 2008, p. 222-231 (cf. p. 65).

[MR91] David A. MCALLESTER et David ROSENBLITT. “Systematic nonlinear planning”. In : *AAAI Conference on Artificial Intelligence*. 1991, p. 634-639 (cf. p. 102, 106).

[McG+08] Conor MCGANN, Frederic PY, Kanna RAJAN, Hans THOMAS, Richard HENTHORN et Rob MCEWEN. “A deliberative architecture for AUV control”. In : *International Conference on Robotics and Automation (ICRA)*. IEEE. 2008, p. 1049-1054 (cf. p. 28).

[MP16] James MCMAHON et Erion PLAKU. “Mission and motion planning for autonomous underwater vehicles operating in spatially and temporally complex environments”. In : *Journal of Oceanic Engineering* 41.4 (2016), p. 893-912 (cf. p. 35).

[MJC14] Alexandre MENIF, Éric JACOPIN et Tristan CAZENAVE. “SHPE : HTN planning for video games”. In : *Computer Games : Third Workshop on Computer Games*. Springer. 2014, p. 119-132 (cf. p. 65).

[Mig03] Ian MIGUEL. *Dynamic flexible constraint satisfaction and its application to AI planning*. Springer Science & Business Media, 2003 (cf. p. 25).

[Mil22] Antoine MILOT. “Algorithmes et architecture pour le contrôle de l’exploration d’une zone par une flotte de sous-marins autonomes”. Thèse de doct. INSA Toulouse, 2022 (cf. p. 34).

[Mit98] Melanie MITCHELL. *An introduction to genetic algorithms*. MIT press, 1998 (cf. p. 28, 29).

[Mon+07] Vincent MONTREUIL, Aurélie CLODIC, Maxime RANSAN et Rachid ALAMI. “Planning human centered robot activities”. In : *International Conference on Systems, Man and Cybernetics (SMC)*. IEEE. 2007, p. 2618-2623 (cf. p. 62).

[MBM16] Christian MUISE, Christopher BECK et Sheila MCILRAITH. “Optimal partial-order plan relaxation via MaxSAT”. In : *Journal of Artificial Intelligence Research (JAIR)* 57 (2016), p. 113-149 (cf. p. 130).

[Nau+03] Dana NAU, Tsz-Chiu AU, Okhtay ILGHAMI, Ugur KUTER, William MURDOCK, Dan WU et Fusun YAMAN. “SHOP2 : An HTN planning system”. In : *Journal of Artificial Intelligence Research (JAIR)* 20 (2003), p. 379-404 (cf. p. 63, 65).

[Nau+21] Dana NAU, Yash BANSOD, Sunandita PATRA, Mark ROBERTS et Ruoxi LI. “GTPyhop : A hierarchical goal+ task planner implemented in Python”. In : *ICAPS Workshop on Hierarchical Planning (HPlan)* (2021), p. 21-25 (cf. p. 62).

[Nie14] Axel NIESTLÉ. *German U-boat losses during World War II : details of destruction*. Frontline Books, 2014 (cf. p. 7).

[Nil09] Nils NILSSON. *The quest for artificial intelligence*. Cambridge University Press, 2009 (cf. p. 30).

[OB15] Santiago ONTANÓN et Michael BURO. “Adversarial hierarchical-task network planning for complex real-time games”. In : *International Joint Conferences on Artificial Intelligence (IJCAI)*. 2015, p. 1652-1658 (cf. p. 64).

[Pat+19] Sunandita PATRA, Malik GHALLAB, Dana NAU et Paolo TRAVERSO. “Interleaving acting and planning using operational models”. In : *Integrated Acting, Planning and Execution (IntEx)*. 2019, p. 46-54 (cf. p. 64).

[Pat+20] Sunandita PATRA, James MASON, Amit KUMAR, Malik GHALLAB, Paolo TRAVERSO et Dana NAU. “Integrating acting, planning, and learning in hierarchical operational models”. In : *International Conference on Automated Planning and Scheduling (ICAPS)*. T. 30. 2020, p. 478-487 (cf. p. 65).

[Pel+23] Damien PELLIER, Alexandre ALBORE, Humbert FIORINO et Rafael BAILON-RUIZ. “HDDL 2.1 : Towards Defining a Formalism and a Semantics for Temporal HTN Planning”. In : *ICAPS Workshop on Hierarchical Planning (HPlan)*. 2023, p. 49-53 (cf. p. 44, 98, 116, 122).

[PF18] Damien PELLIER et Humbert FIORINO. “PDDL4J : A planning domain description library for Java”. In : *Journal of Experimental & Theoretical Artificial Intelligence* 30.1 (2018), p. 143-176 (cf. p. 116).

[PDG23] Laurent PERRON, Frédéric DIDIER et Steven GAY. “The CP-SAT-LP solver”. In : *CP*. 2023, 3 :1-3 :2 (cf. p. 98, 116, 130).

[Py+17] Frédéric PY, José PINTO, Mónica A SILVA, Tor Arne JOHANSEN, João SOUSA et Kanna RAJAN. “Europtus : A mixed-initiative controller for multi-vehicle oceanographic field experiments”. In : *International Symposium on Experimental Robotics (ISER)*. Springer. 2017, p. 323-340 (cf. p. 35).

[Qui+05] Isabelle QUIDU, Nicolas BURLET, Jean-Philippe MALKASSE et Franck FLORIN. “Automatic classification for MCM systems”. In : *Europe Oceans 2005*. T. 2. IEEE. 2005, p. 844-847 (cf. p. 17).

[QGL23] Félix QUINTON, Christophe GRAND et Charles LESIRE. “Market approaches to the multi-robot task allocation problem : a survey”. In : *Journal of Intelligent & Robotic Systems* 107.2 (2023), p. 29 (cf. p. 34).

[RP12] Kanna RAJAN et Frédéric PY. “T-REX : partitioned inference for AUV mission control”. In : *Further advances in unmanned marine vehicles* (2012), p. 171-199 (cf. p. 35).

[Ram+16] Abdeldjalil RAMOUL, Damien PELLIER, Humbert FIORINO et Sylvie PESTY. “HTN planning approach using fully instantiated problems”. In : *International Conference on Tools with Artificial Intelligence (ICTAI)*. IEEE. 2016, p. 113-120 (cf. p. 65).

[RW10] Silvia RICHTER et Matthias WESTPHAL. “The LAMA planner : Guiding cost-based anytime planning with landmarks”. In : *Journal of Artificial Intelligence Research (JAIR)* 39 (2010), p. 127-177 (cf. p. 65).

[RN10] Stuart RUSSELL et Peter NORVIG. *Artificial intelligence a modern approach*. London, 2010 (cf. p. 27, 65).

[Sac75] Earl SACERDOTI. *The nonlinear nature of plans*. Stanford Research Institute, 1975 (cf. p. 67).

[SBE08] Sanem SARIEL, Tucker BALCH et Nadia ERDOGAN. “Naval Mine Countermeasure Missions”. In : *Robotics & Automation Magazine* 15.1 (2008), p. 45-52 (cf. p. 4).

[Sar07] Sanem SARIEL. “An integrated planning, scheduling and execution framework for multi-robot cooperation and coordination”. Thèse de doct. Istanbul Technical University, 2007 (cf. p. 33).

[Sav84] Martin SAVELSBERGH. “Local search in routing problems with time windows”. In : *Annals of Operations Research* 4 (1984), p. 285-305 (cf. p. 29).

[Sch09] Bernd SCHATTENBERG. “Hybrid planning & scheduling”. Thèse de doct. University of Ulm, Germany, 2009 (cf. p. 112).

[Sch15] Bernd SCHATTENBERG. “Hybrid Planning and Scheduling”. In : *KI - Künstliche Intelligenz* 30 (2015), p. 95-97 (cf. p. 67, 102).

[SBB07] Bernd SCHATTENBERG, Julien BIDOT et Susanne BIUNDO. “On the construction and evaluation of flexible plan-refinement strategies”. In : *German Conference on AI*. 2007, p. 367-381 (cf. p. 112).

[SBD18a] Philipp SCHILLINGER, Mathias BÜRGER et Dimos DIMAROGONAS. “Improving multi-robot behavior using learning-based receding horizon task allocation”. In : *14th Conference on Robotics : Science and Systems*. MIT Press Journals. 2018 (cf. p. 33).

[SBD18b] Philipp SCHILLINGER, Mathias BÜRGER et Dimos DIMAROGONAS. “Simultaneous task allocation and planning for temporal logic goals in heterogeneous multi-robot systems”. In : *International Journal of Robotics Research (IJRR)* 37.7 (2018), p. 818-838 (cf. p. 33).

[Sch21a] Dominik SCHREIBER. “Lifted logic for task networks : TOHTN planner Lilotane in the IPC 2020”. In : *International Planning Competition : planner and domain abstracts (IPC)*. 2021 (cf. p. 63).

[Sch21b] Dominik SCHREIBER. “Lilotane : A lifted SAT-based approach to hierarchical planning”. In : *Journal of Artificial Intelligence Research (JAIR)* 70 (2021), p. 1117-1181 (cf. p. 26).

[Sch+19] Dominik SCHREIBER, Damien PELLIER, Humbert FIORINO et Tomáš BALYO. “Tree-REX : SAT-based tree exploration for efficient and high-quality HTN planning”. In : *International Conference on Automated Planning and Scheduling (ICAPS)*. T. 29. 2019, p. 382-390 (cf. p. 63).

[SS19] Mohit SEWAK et Mohit SEWAK. “Deep Q Network (DQN), Double DQN, and Dueling DQN : A Step Towards General Artificial Intelligence”. In : *Deep Reinforcement Learning : Frontiers of Artificial Intelligence* (2019), p. 95-108 (cf. p. 37).

[SSB12] Yagvalkya SHARMA, Subhash Chandra SAINI et Manisha BHANDHARI. “Comparison of Dijkstra’s shortest path algorithm with genetic algorithm for static and dynamic routing network”. In : *International Journal of Electronics and Computer Science Engineering (IJAECS)* 1.2 (2012), p. 416-425 (cf. p. 30).

[Shi+13] Vikas SHIVASHANKAR, Ron ALFORD, Ugur KUTER et Dana NAU. “The GoDeL planning system : a more perfect union of domain-independent and hierarchical planning”. In : *International Joint Conferences on Artificial Intelligence (IJCAI)*. 2013, p. 5-8 (cf. p. 65).

[Shi+16] Vikas SHIVASHANKAR, Ron ALFORD, Mark ROBERTS et David AHA. “Cost-optimal algorithms for hierarchical goal network planning : A preliminary report”. In : *Workshop on Heuristics and Search for Domain Independent Planning (HSDIP)*. 2016, p. 102-111 (cf. p. 65).

[Shi+12] Vikas SHIVASHANKAR, Ugur KUTER, Dana NAU et Ron ALFORD. “A hierarchical goal-based formalism and algorithm for single-agent planning”. In : *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. T. 2. 2012, p. 981-988 (cf. p. 62).

[Sil+16] David SILVER, Aja HUANG, Chris MADDISON, Arthur GUEZ, Laurent SIFRE, George VAN DEN DRIESSCHE, Julian SCHRITTWIESER, Ioannis ANTONOGLOU, Veda PANNEERSHELVAM et Marc LANCTOT. “Mastering the game of Go with deep neural networks and tree search”. In : *Nature* 529.7587 (2016), p. 484-489 (cf. p. 37).

[Sim08] Dan SIMON. “Biogeography-based optimization”. In : *Transactions on Evolutionary Computation* 12.6 (2008), p. 702-713 (cf. p. 29).

[Sim+11] Simona SIMONCELLI, Nadia PINARDI, Paolo ODDO, Arthur MARIANO, Giuseppe MONTANARI, Attilio RINALDI et Marco DESERTI. “Coastal rapid environmental assessment in the Northern Adriatic Sea”. In : *Dynamics of atmospheres and oceans* 52.1-2 (2011), p. 250-283 (cf. p. 13).

[SFC08] David SMITH, Jeremy FRANK et William CUSHING. “The ANML language”. In : *Workshop on Knowledge Engineering for Planning and Scheduling (KEPS)*. T. 31. 2008 (cf. p. 43, 44).

[Sni06] Moshe SNIEDOVICH. “Dijkstra’s algorithm revisited : the dynamic programming connexion”. In : *Control and Cybernetics* 35.3 (2006), p. 599-620 (cf. p. 30).

[SBM09] Shirin SOHRABI, Jorge BAIER et Sheila MCILRAITH. “HTN planning with preferences”. In : *International Joint Conferences on Artificial Intelligence (IJCAI)*. 2009, p. 1790-1797 (cf. p. 64).

[SL10] Christopher SOTZING et David LANE. “Improving the coordination efficiency of limited-communication multi-autonomus underwater vehicle operations using a multiagent architecture”. In : *Journal of Field Robotics* 27.4 (2010), p. 412-429 (cf. p. 33).

[Sto+15] Sebastian STOCK, Masoumeh MANSOURI, Federico PECORA et Joachim HERTZBERG. "Online task merging with a hierarchical hybrid task planner for mobile service robots". In : *International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2015, p. 6459-6464 (cf. p. 69).

[Sze20] Christian SZEGEDY. "A promising path towards autoformalization and general artificial intelligence". In : *Intelligent Computer Mathematics : 13th International Conference (CICM)*. Springer. 2020, p. 3-20 (cf. p. 37).

[Tat77] Austin TATE. "Generating project networks". In : *International Joint Conferences on Artificial Intelligence (IJCAI)*. 1977, p. 888-893 (cf. p. 67).

[TDK94] Austin TATE, Brian DRABBLE et Richard KIRBY. "O-Plan2 : an open architecture for command, planning and control". In : *Intelligent scheduling* 1 (1994), p. 213-239 (cf. p. 67).

[TV14] Paolo TOTH et Daniele VIGO. *Vehicle routing : problems, methods, and applications*. SIAM, 2014 (cf. p. 25).

[Val+15] Mauro VALLATI, Lukas CHRPA, Marek GRZES, Thomas MCCLUSKEY, Mark ROBERTS et Scott SANNER. "The 2014 International Planning Competition : Progress and trends". In : *AI Magazine* 36 (2015), p. 90-98 (cf. p. 40, 129).

[Wil90] David WILKINS. "Can AI planners solve practical problems ?" In : *Computational intelligence* 6.4 (1990), p. 232-246 (cf. p. 67).

[Wil84] David WILKINS. "Domain-independent planning representation and plan generation". In : *Artificial Intelligence* 22.3 (1984), p. 269-301 (cf. p. 67).

[WC18] David WINER et Rogelio CARDONA-RIVERA. "A depth-balanced approach to decompositional planning for problems where hierarchical depth is requested". In : *International Conference on Automated Planning and Scheduling (ICAPS)*. 2018, p. 1-8 (cf. p. 67).

[Woo+05] Ian WOODROW, Chris PURRY, Adam MAWBY et James GOODWIN. "Autonomous AUV mission planning and replanning-towards true autonomy". In : *14th International Symposium on Unmanned Untethered Submersible Technology*. 2005 (cf. p. 34).

[Yil+08] Namik Kemal YILMAZ, Constantinos EVANGELINOS, Pierre LERMUSIAUX et Nicholas PATRIKALAKIS. "Path planning of autonomous underwater vehicles for adaptive sampling using mixed integer linear programming". In : *Journal of Oceanic Engineering* 33.4 (2008), p. 522-537 (cf. p. 28).

[YS03] Håkan YOUNES et Reid SIMMONS. "VHPOP : Versatile heuristic partial order planner". In : *Journal of Artificial Intelligence Research (JAIR)* 20 (2003), p. 405-430 (cf. p. 68).

[ZWJ15] Yudong ZHANG, Shuihua WANG et Genlin JI. "A comprehensive survey on particle swarm optimization algorithm and its applications". In : *Mathematical problems in engineering* (2015), p. 1-38 (cf. p. 29).

A

```
(define
  (domain MCMMission)
  (:requirements
      :durative-actions
      :equality
      :negative-preconditions
      :method-preconditions
      :typing
      :numeric-fluents
      :timed-initial-literals
      :hierarchy)

  (:types
    MainShip Deployable - Ship
    AUV USV - Deployable
    A18 A9 - AUV
    Area
  )

  (:predicates
    (at ?ship - Ship ?area - Area)
    (on ?ship1 ?ship2 - Ship)
    (launched ?ship - Ship)
    (authorized_on ?auv - Deployable ?area - Area)
    (busy ?ship - Ship)
  )

  (:functions
    (travel-time ?dep - Deployable ?area_from ?area_to - Area)
    (patern_duration ?auv - AUV ?area - Area)
    (launching_time ?dep - Deployable ?ship - Ship)
    (recovery_time ?dep - Deployable ?ship - Ship)
    (GPS_Fix_Time ?auv - AUV)
    )

  (:task scan_area :parameters (?area - Area))
  (:task get_to :parameters (?area - Area ?ship - Ship))
  (:task recover :parameters (?ship - Ship))
  (:task launch :parameters (?dep - Deployable))
  (:task dummy :parameters ())
```

```
;Scanning method ------------------------------------
(:method scan_area
  :parameters (?area - Area ?auv - AUV)
  :task (scan_area ?area)
  :ordered-subtasks (and
    (task0 (get_to ?area ?auv))
    (task1 (GPS_Fix ?auv ?area))
    (task2 (apply_scanning_patern ?area ?auv))
    )
)

;Movement methods ------------------------------------
(:method get_to_AUV_launched
  :parameters (?area_from ?area_to - Area ?auv - AUV)
  :task (get_to ?area_to ?auv)
  :precondition (and
    (at ?auv ?area_from)
    (launched ?auv)
    )
  :subtasks (and
    (task0 (move_deployable ?auv ?area_from ?area_to))
  )
)

(:method get_to_AUV_launching
  :parameters (?area_from ?area_to - Area ?auv - AUV ?ship - Ship)
  :task (get_to ?area_to ?auv)
  :precondition (and
    (on ?auv ?ship)
    (launched ?ship)
    )
  :ordered-subtasks (and
    (task0 (launch_ship ?auv ?ship ?area_from))
    (task1 (move_deployable ?auv ?area_from ?area_to))
  )
)

(:method get_to_AUV_via_USV
  :parameters (?area_from ?area_to - Area ?auv - AUV ?usv - USV)
  :task (get_to ?area_to ?auv)
  :precondition (and
    (at ?auv ?area_from)
    (launched ?auv)
    (launched ?usv)
    )
  :ordered-subtasks (and
    (task0 (get_to ?area_from ?usv))
    (task1 (recover_ship ?auv ?usv ?area_from))
    (task2 (move_deployable ?usv ?area_from ?area_to))
    (task3 (launch_ship ?auv ?usv ?area_to))
    )
)

(:method get_to_USV_launched
  :parameters (?area_from ?area_to - Area ?usv - USV)
  :task (get_to ?area_to ?usv)
  :precondition (and
    (at ?usv ?area_from)
    (launched ?usv)
    )
  :subtasks (and
    (task0 (move_deployable ?usv ?area_from ?area_to))
  )
)
```

```
(:method get_to_USV_launching
  :parameters (?area_from ?area_to - Area ?usv - USV
      ?mainship - MainShip)
  :task (get_to ?area_to ?usv)
  :precondition (and
    (on ?usv ?mainship)
    (at ?mainship ?area_from)
    )
  :ordered-subtasks (and
    (task0 (launch_ship ?usv ?mainship ?area_from))
    (task1 (move_deployable ?usv ?area_from ?area_to))
  )
)

(:method get_to_AUV_noop
  :parameters (?area - Area ?auv - AUV)
  :task (get_to ?area ?auv)
  :precondition (and
    (at ?auv ?area)
    )
  :subtasks()
)

(:method get_to_USV_noop
  :parameters (?area - Area ?usv - USV)
  :task (get_to ?area ?usv)
  :precondition (and
    (at ?usv ?area)
    )
  :subtasks()
)

;Recovery methods -------------------------------------
(:method recover_on_Main_Ship
  :parameters (?dep - Deployable ?mainship - MainShip
      ?area - Area)
  :task (recover ?dep)
  :precondition (and
    (at ?mainShip ?area)
    )
  :ordered-subtasks (and
    (task0 (get_to ?area ?dep))
    (task1 (recover_ship ?dep ?mainship ?area))
    )
)

(:method recover_AUV_via_USV
  :parameters (?auv - AUV ?usv - USV ?area - Area)
  :task (recover ?auv)
  :subtasks (and
    (task0 (get_to ?area ?usv))
    (task1 (get_to ?area ?auv))
    (task2 (recover_ship ?auv ?usv ?area))
    )
  :ordering  (and
    (< task0 task2)
    (< task1 task2)
    )
)
```

```
;Movement Actions ------------------------------------
(:durative-action move_deployable
  :parameters (?dep - Deployable ?area_from
      ?area_to - Area)
  :duration (= ?duration (travel-time ?dep ?area_from ?area_to))
  :condition (and
    (at start (at ?dep ?area_from))
    (over all (authorized_on ?dep ?area_from))
    (over all (authorized_on ?dep ?area_to))
    (at start (launched ?dep))
    (over all (launched ?dep))
    )
  :effect (and
    (at start (not (at ?dep ?area_from)))
    (at end (at ?dep ?area_to))
  )
)

(:durative-action move_mainShip
  :parameters (?mainShip - MainShip ?area_from ?area_to - Area)
  :duration (= ?duration 1)
  :condition (and
    (at start (at ?mainShip ?area_from))
    (at start (launched ?mainShip))
    (over all (launched ?mainShip))
    )
  :effect (and
    (at start (not (at ?mainShip ?area_from)))
    (at end (at ?mainShip ?area_to))
  )
)

(:durative-action apply_scanning_patern
  :parameters (?area - Area ?auv - AUV)
  :duration (= ?duration (patern_duration ?auv ?area))
  :condition (and
    (over all (authorized_on ?auv ?area))
    (over all (at ?auv ?area))
    )
  :effect (and
    (at start (busy ?auv))
    (at end (not (busy ?auv)))
  )
)

(:durative-action launch_ship
  :parameters (?small_ship - Deployable ?big_ship - Ship
      ?area - Area)
  :duration (= ?duration (launching_time ?small_ship ?big_ship))
  :condition (and
    (at start (on ?small_ship ?big_ship))
    (at start (at ?big_ship ?area))
    (at start (not (busy ?big_ship)))
    )
  :effect (and
    (at start (busy ?big_ship))
    (at end (not (busy ?big_ship)))
    (at start (not (on ?small_ship ?big_ship)))
    (at end (at ?small_ship ?area))
    (at end (launched ?small_ship))
    )
)
```

```
226     (:durative-action recover_ship
227       :parameters (?small_ship - Deployable ?big_ship - Ship
228           ?area - Area)
229       :duration (= ?duration (recovery_time ?small_ship ?big_ship))
230       :condition (and
231         (at start (at ?big_ship ?area))
232         (at start (at ?small_ship ?area))
233         (over all (at ?big_ship ?area))
234         (over all (at ?small_ship ?area))
235         )
236       :effect (and
237         (at end (not (launched ?small_ship)))
238         (at end (on ?small_ship ?big_ship))
239         )
240     )
241
242     (:durative-action GPS_Fix
243       :parameters (?auv - AUV ?area - Area)
244       :duration (= ?duration (GPS_Fix_Time ?auv))
245       :condition (and
246         (at start (at ?auv ?area))
247         (over all (at ?auv ?area))
248         (at end (at ?auv ?area))
249         )
250       :effect (and
251         (at start (busy ?auv))
252         (at end (not (busy ?auv)))
253       )
254     )
255   )
```

Listing A.1 : Fichier HDDL définissant le domaine associé à la chasse aux mines.

```
(define (problem mcmv1auv1)
 (:domain MCMMission)
 (:objects
   mainShip - MainShip
   auv1 - AUV
   area1 area2 area3 - Area
 )

 (:htn
   :parameters ()
   :subtasks (and
     (task0 (scan_area area2))
     (task1 (scan_area area3))
     (task2 (recover auv1))
     )
   :ordering (and
     (< task0 task2)
     (< task1 task2)
     )
 )

 (:init
   (at mainShip area1)
   (launched mainShip)

   (on auv1 mainShip)

   (authorized_on auv1 area1)
   (authorized_on auv1 area2)
   (authorized_on auv1 area3)

   (= (travel-time auv1 area1 area1) 1.0)
   (= (travel-time auv1 area1 area2) 23.4)
   (= (travel-time auv1 area1 area3) 28.4)
   (= (travel-time auv1 area2 area1) 20.1)
   (= (travel-time auv1 area2 area2) 1.0)
   (= (travel-time auv1 area2 area3) 12.6)
   (= (travel-time auv1 area3 area1) 28.4)
   (= (travel-time auv1 area3 area2) 12.6)
   (= (travel-time auv1 area3 area3) 1.0)

   (= (patern_duration auv1 area1) 208.2)
   (= (patern_duration auv1 area2) 245.2)
   (= (patern_duration auv1 area3) 453.1)

   (= (launching_time auv1 mainShip) 50.0)

   (= (recovery_time auv1 mainShip) 67.0)

   (= (GPS_Fix_Time auv1) 6.2)
 )
)
```

Listing A.2 : Fichier HDDL définissant une instance de problème de chasse aux mines.

B

Les Figures B.1-B.6 représentent différentes itérations successives de CPFD appliquées à un réseau de tâches initial contenant une unique tâche $T0$. Sur chacune des figures, les tâches avec une lettre majuscule sont considérées comme des tâches abstraites et celles avec une minuscule comme des tâches primitives :

— **Figure B.1 :** À chaque itération, CPFD tente de résoudre l'une des tâches sans prédécesseurs du réseau. Ici, $T0$ est l'unique tâche du réseau, elle est donc sélectionnée par CPFD que essaie de la décomposer selon l'une de ses méthodes.

— **Figure B.2 :** Une fois la tâche $T0$ résolue et décomposée, deux tâches sont sans prédécesseurs :$t4$ et $t2$. Supposons que $t4$ ait été non déterministiquement sélectionnée par CPFD et que l'action correspondante, a_4 soit applicable à I.

— **Figure B.3** : L'action a_4 a été introduite dans le layer courant par la résolution de $t4$. La tâche $t4$ a été marquée comme résolue dans le réseau de tâches. Enfin une autre tâche sans prédécesseurs est sélectionnée par CPFD. Ici, seule $t2$ est disponible.

— **Figure B.5** : Si a_2 l'action résolvant $t2$ n'est pas mutuellement avec a_4 ou que ses préconditions ne sont pas satisfaites dans I, $t4$ ne peut pas être résolue et CPFD introduit un nouveau layer au plan solution en appliquant les effets de π_1 à I. Il en résulte un nouvel état à partir duquel le prochain layer sera construit.

— **Figure B.4** : Si au contraire a_4 et a_2 sont mutuellement indépendantes, et que les préconditions de a_2 sont vérifiées dans I, a_2 est introduite dans le layer courant et $t2$ est marquée comme résolue.

— **Figure B.6** : Une fois toutes les tâches sans prédécesseurs résolues, CPFD les retire du réseau de tâches et construit le layer suivant à partir de l'état $\gamma(I, \pi_i)$.

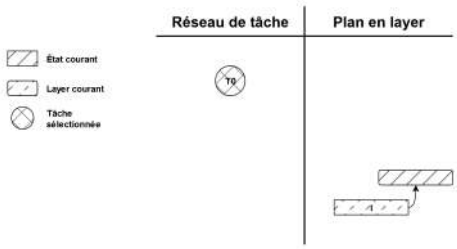

Figure B.1. : Le réseau de tâches initial est composé d'une unique tâche $T0$, elle est donc la seule à pouvoir être sélectionnée.

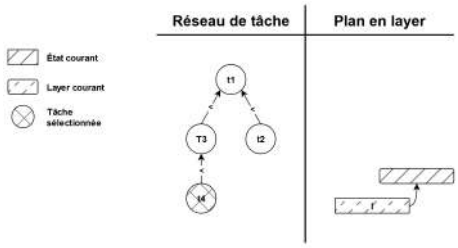

Figure B.2. : Une tâche sans prédécesseur est sélectionnée pour être résolue. Dans le cas présent, $t4$ a été sélectionnée parmi $\{t4, t2\}$.

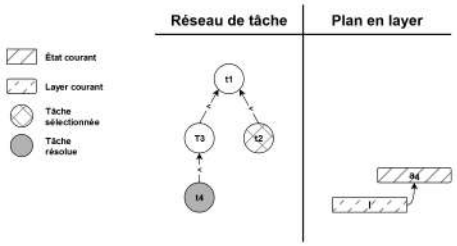

Figure B.3. : La tâche $t4$ est résolue en insérant a_4 dans le layer courant, puis seule $t2$ peut être sélectionnée.

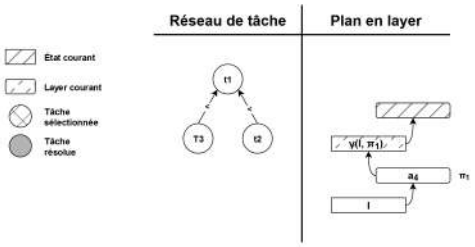

Figure B.4. : Si a_2 ne peut pas être résolue dans le layer courant, CPFD introduit un nouveau layer en appliquant les effets du layer construit et en retirant du réseau les tâches résolues.

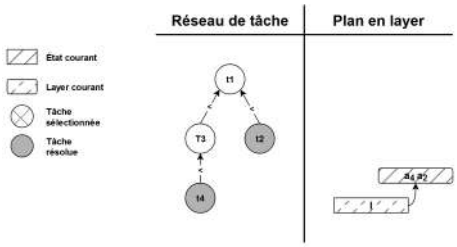

Figure B.5. : Si a_2 et a_4 sont mutuellement indépendantes, $t2$ peut être résolue en insérant a_2 au layer en courant.

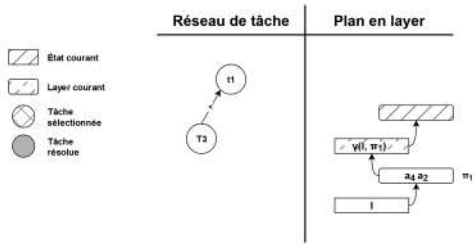

Figure B.6. : Lorsque toute les tâches sans prédécesseurs sont résolues, le layer courant est appliqué à l'état courant, les tâches résolues sont retirées et un nouveau layer introduit.