Planification par satisfaction de bases de clauses

Frédéric Maris[1] and Pierre Régnier[1] and Vincent Vidal[2]

[1] IRIT – Université Paul Sabatier
Toulouse, France
{maris,regnier}@irit.fr

[2] CRIL – Université d'artois
Lens, France
vidal@cril.univ-artois.fr

L'amélioration considérable de la technologie et des performances des prouveurs SAT a rendu possible leur utilisation pour la résolution de différents problèmes d'intelligence artificielle, et parmi ceux-ci celui de la génération de plans d'actions. Nous présentons dans ce chapitre une formalisation des principaux codages des problèmes de planification de type STRIPS sous forme de bases de clauses, inspirés des paradigmes classiques de résolution de problèmes de planification, assortie d'une étude bibliographique détaillée des principaux travaux concernant la planification par satisfaction de bases de clauses.

1 Introduction

La planification – discipline de l'intelligence artificielle – cherche à concevoir des systèmes capables de générer automatiquement, grâce à une procédure formalisée, un résultat articulé, sous la forme d'un système intégré de décisions appelé plan solution. Ce dernier est une collection organisée de descriptions d'opérations ; il est destiné à guider l'action d'un ou plusieurs agents exécuteurs (systèmes robotiques ou humains) qui doivent agir dans un monde particulier pour atteindre un but préalablement défini. L'exécution est la réalisation d'actions effectuée suivant les directives du plan. Elle vise à réaliser la prédiction que constitue ce plan ; lorsqu'elle est conforme à ce qu'il indique, elle doit permettre (en l'absence d'événements imprévus et si la modélisation du monde est pertinente) de faire évoluer l'univers de l'état initial vers un état satisfaisant le but.

Depuis le système fondateur STRIPS [25], et dans le cadre « classique »de la planification qu'il a délimité – environnement statique, observabilité totale, agent omniscient, actions atomiques et déterministes – de très gros progrès ont été réalisés. Il y a encore peu, depuis [16] jusqu'à [33], l'approche de la planification qui était l'objet essentiel des recherches était fondée sur les stratégies de raffinement dans les espaces de plans partiels. Au sein de la communauté, les autres approches étaient marginalisées. En 1995 cependant, l'apparition du planificateur GRAPHPLAN [7, 8] fut à l'origine d'un bouleversement de cet ordre bien établi : ses performances et les idées qui guidaient sa conception ont inspiré un grand nombre de travaux. Ceux-ci ont permis aux algorithmes de planification STRIPS d'augmenter leurs performances de manière si importante que l'on peut maintenant envisager des applications réelles à moyen terme. Actuellement, certains de ces nouveaux algorithmes, grâce à l'utilisation d'heuristiques inspirées par GRAPHPLAN, permettent de résoudre rapidement des problèmes complexes qui, même s'ils sont issus de benchmarks

artificiels, sont largement hors de portée de l'humain. Les plans solutions générés, bien que non optimaux, peuvent comporter plusieurs milliers d'actions.

Après avoir décrit brièvement les principaux paradigmes de résolution de problèmes de planification (Section 2), nous donnons quelques définitions préliminaires indispensables à la compréhension des codages de planification (Section 3). Nous présentons ensuite une formalisation des différents codages, suivant les techniques classiques de planification dont ils s'inspirent : planification dans les espaces d'états (Section 4), planification dans les espaces de plans (Section 5), codage du graphe de planification (Section 6). Ces codages sont souvent décrits de façon informelle dans la littérature : nous en donnons une expression formelle et concise, qui intègre de plus un certain nombre d'améliorations par rapport aux codages originaux que nous ne détaillerons pas ici (voir [70, 50] pour plus de détails sur ces améliorations). Nous terminons par une étude bibliographique détaillée des principaux travaux concernant la planification SAT (Section 7).

2 Planification strips : principaux paradigmes de résolution

2.1 Recherche dans les espaces d'états

En planification par recherche dans les espaces d'états, les nœuds du graphe de recherche représentent les états successifs du monde et les arcs, les actions qui permettent de passer d'un état à un autre. L'algorithme tente de construire un chemin (plan solution) qui permette de passer de l'état initial du problème à un état but. Il se termine quand l'état courant contient les buts à atteindre. Parmi les planificateurs actuels les plus performants qui sont basés sur cette technique, on trouve HSP et HSPr [9], FF [30], ALTALT [56], YAHSP [70], SGPLAN [17], DOWNWARD [28] et de nombreux autres. Ils calculent une heuristique généralement très informative et le plus souvent non admissible inspirée du graphe de planification de GRAPHPLAN [7, 8] qui leur permet de guider la recherche de manière très efficace. En contrepartie, ils n'offrent le plus souvent pas de garantie sur l'optimalité en nombre d'actions des plans solutions. De plus, même si un post-traitement permet de les paralléliser [58, 3], ces plans demeurent généralement moins flexibles que ceux obtenus par d'autres approches (la flexibilité d'un plan représentant la possibilité de pouvoir, dans une certaine mesure, modifier la date de début des actions ou l'allocation des ressources sans changer son résultat).

2.2 Recherche dans les espaces de plans

En planification par recherche dans les espaces de plans partiels, les nœuds du graphe de recherche représentent des plans partiels et les arcs, les opérations d'extension de ces plans. L'algorithme cherche à étendre un plan initial qui représente le problème posé pour obtenir un plan solution. Il se termine lorsque toutes les préconditions de toutes les actions présentes dans le plan courant sont produites par d'autres actions du plan et que ce dernier ne comporte plus aucune action qui puisse potentiellement interférer avec une autre. Cette technique, introduite par [16], a été l'objet de très nombreux travaux [73, 33]. Elle fut délaissée après l'arrivée de GRAPHPLAN [7, 8], de SATPLAN [36], de BLACKBOX [39, 41] et des planificateurs par recherche heuristique dans les espaces d'états. Elle connaît actuellement un renouveau avec les planificateurs REPOP [55], VHPOP [77] et CPT [71, 72] qui utilisent des heuristiques basées sur le calcul du graphe de planification de GRAPHPLAN pour guider efficacement la recherche. Cette approche demeure ainsi intéressante car elle permet d'obtenir des plans possédant un haut degré de flexibilité.

2.3 Recherche dans les graphes de planification

L'algorithme de GRAPHPLAN [7, 8] et des planificateurs qui en dérivent comme IPP [43], STAN [26] ou LCGP [15, 68], consiste à développer d'abord, niveau par niveau et en temps polynomial (fonction du nombre total de variables d'état possibles pour la représentation de l'univers, de celles de l'état initial, et du nombre d'actions), un espace de recherche compact appelé graphe de planification (voir Section 6). Pour cette phase (dite de construction), il n'utilise pas toutes les informations indispensables à l'obtention d'un plan solution qui sont prises en compte au fur et à mesure par les autres approches (exclusions mutuelles entre variables d'état ou actions). Ces contraintes sont simplement calculées et enregistrées à chaque niveau du graphe sous forme d'exclusions mutuelles à la manière des CSP [32]. Cet espace de recherche se développe donc plus facilement, mais, en contrepartie, son achèvement ne coïncide plus avec l'obtention d'un plan solution qu'une deuxième phase (dite d'extraction) cherche alors à extraire à partir du graphe de planification et des contraintes enregistrées. Les plans ainsi générés peuvent être représentés par des séquences d'ensembles d'actions indépendantes (voir Section 3), chaque ensemble correspondant à un niveau du graphe de planification. Cette approche permet de générer des plans optimaux en nombre de niveaux et possédant une bonne flexibilité. Celle-ci reste généralement moins bonne que celle obtenue par recherche dans les espaces de plans partiels [55].

2.4 Planification par satisfaction de bases de clauses

En 1992, les approches classiques de la planification, basées sur la recherche dans les espaces de plans partiels et dans les espaces d'états, n'offraient pas de performances convainquantes. Kautz et Selman [37] ont alors proposé, avec l'approche de la planification par satisfaction de bases de clauses (planification SAT) implémentée dans le planificateur SATPLAN, de profiter des progrès constants effectués dans le domaine des techniques SAT pour résoudre efficacement les problèmes de planification. L'utilisation de ces techniques offre également un cadre formel permettant l'utilisation naturelle de connaissances sur le domaine, connaissances représentables sous forme de clauses exprimant des contraintes entre actions et fluents. Les plans solutions potentiels au problème de planification posé peuvent être représentés par différents codages dont chacun est une manière d'appréhender la structure de ces plans (codages dans les espaces d'états et dans les espaces de plans, codage du graphe de planification). La traduction du problème, par le biais d'un codage donné, fournit une base de clauses qui est ensuite donnée en entrée à un prouveur SAT. Celui-ci cherche alors un modèle de cette base de clauses. La traduction inverse du modèle trouvé par le prouveur (lorsqu'il existe), fournit un plan solution au problème initial.

Comme les approches de la planification SAT travaillent sur un ensemble fini de variables propositionnelles et que deux actions identiques peuvent apparaître à des endroits différents d'un même plan, on les différencie en leur associant des propositions différentes. La longueur d'un plan solution ne pouvant être connue à l'avance, on ne peut créer un codage unique permettant de résoudre le problème posé (il faudrait une infinité de propositions représentant toutes les actions de tous les plans possibles). Ce point est confirmé par la complexité théorique de la planification classique propositionnelle, qui est PSPACE-difficile [11], alors que la planification sous horizon bornée est NP-difficile. On emploie donc un codage représentant tous les plans d'une longueur fixée en commençant à une borne minimale de l'horizon (soit 0, soit une borne obtenue par une approximation comme la construction d'un graphe de planification). Tant qu'aucun plan solution n'est trouvé par la résolution de ce codage, cette borne est augmentée. Les plans solutions correspondent aux modèles de la base de clauses issue du codage ; ils sont donc optimaux par rapport à la borne courante de l'horizon. Les approches de la planification SAT donnent

des planificateurs complets puisque s'il existe une solution, ils la trouvent ; par contre, le problème de l'arrêt lorsqu'il n'y a pas de solution n'est pas résolu.

3 Définitions préliminaires

Nous nous plaçons dans le cadre de la planification STRIPS propositionnelle, ne faisant intervenir que des actions définies à l'aide d'un ensemble fini de symboles propositionnels atomiques. Un problème STRIPS propositionnel peut éventuellement être obtenu en instanciant des schémas d'opérateurs décrits dans un langage plus riche, par exemple PDDL [54] (≪ Planning Domain Definition Language ≫), à l'aide des constantes définies pour un problème particulier.

Définition 1 *Un* état *est un ensemble fini de symboles propositionnels aussi appelés* fluents*.*

Définition 2 *Une* action *a est un triplet $\langle pr, ad, de\rangle$ où pr, ad et de dénotent des ensembles finis de fluents.* $\mathrm{Prec}(a)$, $\mathrm{Add}(a)$, $\mathrm{Del}(a)$ *dénotent respectivement les ensembles pr, ad, de et représentent les* préconditions, ajouts *et* retraits *de a.*

Définition 3 *Un* problème de planification Π *est un quadruplet $\langle F, A, I, B\rangle$ où F dénote un ensemble fini de fluents, A dénote un ensemble fini d'actions construites à partir des fluents de F, $I \subseteq F$ dénote un ensemble fini de fluents qui représentent l'*état initial *du problème, et $B \subseteq F$ dénote un ensemble fini de fluents qui représentent les* buts *du problème.*

Un objectif de la planification SAT est de calculer des plans parallèles minimaux en nombre de niveaux d'actions parallèles. Nous définissons donc la notion de plan parallèle, qui généralise la notion de plan séquentiel.

Définition 4 *Un* plan parallèle *est une séquence finie d'ensembles finis d'actions. Le plan vide est noté $\langle\rangle$, et un plan non vide est noté $\langle Q_1, \ldots, Q_n\rangle$, avec $n \in \mathbb{N}^*$. Par abus de langage, un plan $\langle Q_1, \ldots Q_n\rangle$ où $n = 0$ dénotera le plan vide. Pour un ensemble d'actions Q_i, i dénote le* niveau *de Q_i dans le plan. L'ensemble des plans que l'on peut construire à partir d'un ensemble d'actions A est noté $(2^A)^*$.*

Définition 5 *Soient F un ensemble fini de fluents et A un ensemble fini d'actions. L'application $\uparrow : 2^F \times 2^A \to 2^F$ est définie par $E \uparrow Q = \left(E \setminus \bigcup_{a\in Q} \mathrm{Del}(a)\right) \cup \bigcup_{a\in Q} \mathrm{Add}(a)$.*

L'application d'un plan parallèle P à un état E fournit soit un état, soit $\perp$ en cas d'échec. Elle consiste à appliquer chaque ensemble d'actions Q du plan sur le résultat de l'application des ensembles d'actions qui précèdent Q dans P, en prenant E comme état de départ. L'application d'un plan échoue si l'union des préconditions des actions de Q ne sont pas satisfaites dans l'état auquel on l'applique : le résultat sera alors $\perp$.

Définition 6 *Soient F un ensemble de fluents et A un ensemble d'actions. L'application $\Re : (2^F \cup \{\perp\}) \times (2^A)^* \to (2^F \cup \{\perp\})$ est définie par :*

$$E\Re\langle Q_1, Q_2, \ldots, Q_n\rangle = \begin{cases} (E \uparrow Q_1)\ \Re\ \langle Q_2, \ldots, Q_n\rangle & \text{si } n \neq 0, E \neq \perp \text{ et} \\ & \bigcup_{a\in Q_1} \mathrm{Prec}(a) \subseteq E \\ E & \text{si } n = 0 \\ \perp & \text{sinon} \end{cases}$$

La contrainte essentielle que doit vérifier un plan parallèle pour être un plan solution est la suivante : l'exécution d'un plan doit conduire au même état résultant, quel que soit l'ordre d'exécution que l'on pourrait imposer aux actions de chacun de ses ensembles. Pour obtenir ce résultat avec l'utilisation de descriptions d'action de type STRIPS, toutes les actions d'un même ensemble d'actions d'un plan doivent être indépendantes deux à deux, c'est-à-dire que leurs effets ne doivent pas être contradictoires (une action ne doit pas retirer un ajout d'une autre action) et elles ne doivent pas interférer (une action ne doit pas retirer une précondition d'une autre action). On peut montrer qu'une relation moins contrainte que celle-ci peut être utilisée : la relation d'autorisation [68, 15, 50]. Elle permet d'améliorer notablement les performances de GRAPHPLAN et de réduire la taille du codage du problème en une base de clauses, mais fait perdre l'optimalité du plan en nombre de niveaux (au sens classique, par rapport à l'indépendance).

Définition 7 *Deux actions a et b sont* effets-indépendantes *(noté $a\|_e b$) si et seulement si elles n'ont pas d'effets contradictoires :* $(\mathrm{Add}(a) \cap \mathrm{Del}(b)) \cup (\mathrm{Add}(b) \cap \mathrm{Del}(a)) = \emptyset$. *Les actions a et b sont* interférence-indépendantes *(noté $a\|_i b$) si et seulement si elles n'interfèrent pas :* $(\mathrm{Prec}(a) \cap \mathrm{Del}(b)) \cup (\mathrm{Prec}(b) \cap \mathrm{Del}(a)) = \emptyset$. *Elles sont* indépendantes *(noté $a\|b$) si et seulement si elles sont effets-indépendantes et interférence-indépendantes. Un ensemble d'actions Q est un* ensemble indépendant *si et seulement si* $\forall \{a, b\} \subseteq Q, a \neq b \Rightarrow a\|b$.

Définition 8 *Soient* $\Pi = \langle F, A, I, B\rangle$ *un problème de planification et* $P = \langle Q_1, \ldots, Q_n\rangle$ *un plan. P est un* plan solution *de* Π *si et seulement si* $\forall i \in [1, n], (Qi \subseteq A$ *et* Q_i *est indépendant) et* $B \subseteq I\Re P$.

Les codages d'un problème de planification sous forme de base de clauses seront formalisés par des règles de réécriture. Ces règles seront exprimées par rapport à un problème de planification $\Pi = \langle F, A, I, B\rangle$ et un entier $k \geq 0$ qui représentera le nombre de niveaux des plans produits. Nous utiliserons également les sous-ensembles de F suivants : $F_p = \bigcup_{a\in A} \mathrm{Prec}(A)$, $F_a = \bigcup_{a\in A} \mathrm{Add}(A)$, $F_d = \bigcup_{a\in A} \mathrm{Del}(A)$ dénotant respectivement les ensembles de fluents se trouvant en précondition, en ajout et en retrait des actions. Les règles des codages seront parfois exprimées sous forme non clausale pour des raisons de compacité, mais la transformation sous forme de clauses est immédiate.

Soient $A = \{a_1, \ldots, a_n\}$ un ensemble fini quelconque et f une règle de réécriture. Pour une action $a \in A$, soit $P[a]$ une propriété que peut (ou non) vérifier a. Soit $P_A = \{a \in A \mid P[a]$ vraie$\} = \{b_1, \ldots, b_m\}$:

- $\bigwedge_{a\in A} f(a)$ dénote $F = f(a_1) \wedge \ldots \wedge f(a_n)$. Si $A = \emptyset$, alors $F = \top$.
- $\bigvee_{a\in A} f(a)$ dénote $F = f(a_1) \vee \ldots \vee f(a_n)$. Si $A = \emptyset$, alors $F = \bot$.
- $\bigwedge_{a\in A \mid P[a]} f(a)$ dénote $F = f(b_1) \wedge \ldots \wedge f(b_m)$. Si $P_A = \emptyset$, alors $F = \top$.
- $\bigvee_{a\in A \mid P[a]} f(a)$ dénote $F = f(b_1) \vee \ldots \vee f(b_m)$. Si $P_A = \emptyset$, alors $F = \bot$.

Pour deux formules F_1 et F_2 et une propriété P, la règle de réécriture « si P est satisfaite, alors F_1 sinon F_2 »sera notée $\left(\; P \hookrightarrow F_1 \mid F_2 \; \right)$.

4 Les codages dans les espaces d'états

Ces codages sont basés sur les transitions entre les niveaux successifs du plan, depuis l'état initial jusqu'au but. Le parallélisme y est codé grâce à la notion d'indépendance entre actions simultanées. Pour conserver les fluents non affectés par les actions du niveau suivant, on code la notion de frame-axiome [52]. Nous décrivons d'abord la technique la

plus efficace en termes de compacité et de temps de résolution [49] qui utilise des frame-axiomes explicatifs. Nous présentons ensuite une variation de ce codage utilisant des actions particulières appelées no-ops [36].

Les règles du codage dans les espaces d'états produisent des propositions de la forme suivante, pour une action $a \in A$ et un fluent $f \in F$:

- $a(i)$ est vraie si et seulement si l'action a est présente au niveau i du plan,
- $f(i)$ est vraie si et seulement si le fluent f est présent après l'exécution successive des actions du plan des niveaux 1 à i.

4.1 Codage dans les espaces d'états avec frame-axiomes explicatifs

Le présence de f au niveau i signifie qu'il est présent après l'application successive de toutes les actions associées à des propositions qui sont vraies, du niveau 1 jusqu'au niveau i. Ce codage comporte cinq règles :

1. *État initial et but :* les fluents de l'état initial sont vrais au niveau 0, ceux qui n'en font pas partie sont faux au niveau 0, et les fluents du but sont vrais au niveau k.

$$\left[\bigwedge_{f \in I} \; f(0)\right] \wedge \left[\bigwedge_{f \in (F \setminus I)} \; \neg f(0)\right] \wedge \left[\bigwedge_{f \in B} \; f(k)\right]$$

2. *Préconditions et effets des actions :* si une action appartient au plan, alors ses préconditions sont vérifiées et ses effets sont produits.

$$\bigwedge_{i \in [1,k]} \; \bigwedge_{a \in A} \; \left[a_i \Rightarrow \left(\bigwedge_{f \in \mathrm{Prec}(a)} \; f(i-1)\right) \wedge \left(\bigwedge_{f \in \mathrm{Add}(a)} \; f(i)\right) \wedge \left(\bigwedge_{f \in \mathrm{Del}(a)} \; \neg f(i)\right)\right]$$

3. *Frame-axiomes explicatifs de retrait :* si un fluent devient faux entre deux niveaux successifs du plan, alors une action au moins qui le retire doit avoir été appliquée. Il faut que le fluent existe à un instant donné, il doit donc appartenir à I ou à F_a. Il lui faut aussi pouvoir être retiré par une action, il doit donc aussi appartenir à F_d.

$$\bigwedge_{i \in [1,k]} \; \bigwedge_{f \in ((I \cup F_a) \cap F_d)} \; \left[f(i-1) \wedge \neg f(i) \Rightarrow \bigvee_{a \in A \,|\, f \in \mathrm{Del}(a)} \; a(i)\right]$$

4. *Frame-axiomes explicatifs d'ajout :* si un fluent devient vrai entre deux niveaux successifs du plan, alors une action au moins qui l'établit doit avoir été appliquée. Il faut que le fluent puisse ne pas exister à un instant donné, donc ne pas appartenir à I ou appartenir à F_d. Il lui faut aussi pouvoir être ajouté par une action donc appartenir à F_a.

$$\bigwedge_{i \in [1,k]} \; \bigwedge_{f \in (((F \setminus I) \cup F_d) \cap F_a)} \; \left[\neg f(i-1) \wedge f(i) \Rightarrow \bigvee_{a \in A \,|\, f \in \mathrm{Add}(a)} \; a(i)\right]$$

5. *Interférences :* deux actions non indépendantes ne peuvent pas être exécutées au même niveau. Les effets contradictoires sont déjà pris en compte par la règle 2 ; il suffit donc d'interdire à un même niveau les actions uniquement interférentes.

$$\bigwedge_{i \in [1,k]} \; \bigwedge_{\{a_m, a_n\} \subseteq A \,|\, m < n \wedge (a_m \|_e a_n) \wedge \neg (a_m \|_i a_n)} \; [\neg a_m(i) \vee \neg a_n(i)]$$

4.2 Codage dans les espaces d'états avec no-ops

L'idée de ce codage vient du planificateur GRAPHPLAN et a été proposée pour la première fois dans [36]. L'ensemble A des actions du problème est augmenté par des actions particulières appelées no-ops qui ont pour unique précondition un fluent de F, pour unique ajout ce même fluent, et n'ont pas de retraits. Les no-ops sont créés pour chaque fluent de F et sont ajoutés à A qui devient donc $A \cup \{\langle\{f\},\{f\},\{\}\rangle \mid f \in F\}$. Ils participeront à la construction du plan solution mais les propositions correspondantes ne seront pas incluses lors du décodage du plan. La présence d'un no-op à un niveau i signifie que le fluent qu'il représente était déjà présent au niveau $i-1$ et sera toujours présent au niveau i. Un no-op est en conflit avec les actions qui retirent le fluent qu'il représente ; ainsi, une clause d'interférence de la règle 5 du codage précédent faisant intervenir un no-op représentera le fait qu'un fluent est : soit conservé d'un niveau sur l'autre (si la variable propositionnelle représentant le no-op est vraie et la variable représentant l'action conflictuelle est fausse), soit retiré (cas inverse). La seule différence avec le codage précédent est le remplacement des règles 3 et 4 portant sur les frame-axiomes explicatifs par la règle suivante :

3-4) *Frame-axiomes avec no-ops :* un fluent vrai au niveau i implique la disjonction des actions du niveau i qui le produisent, y compris son no-op.

$$\bigwedge_{i\in[1,k]} \bigwedge_{f\in(((F\setminus I)\cup F_d)\cap F_a)} \left[f(i) \Rightarrow \bigvee_{a\in A | f\in \mathrm{Add}(a)} a(i) \right]$$

5 Codages dans les espaces de plans

Ces codages ne traduisent plus seulement les transitions qui s'opèrent entre niveaux successifs du plan, mais expriment maintenant les relations de causalité entre les actions qui le constituent. Dans les espaces d'états, l'application des actions est envisagée séquentiellement : une action ne peut apparaître à un niveau du plan que lorsque ses préconditions sont satisfaites au niveau précédent. Dans les espaces de plans, une action apparaît à un niveau du plan parce qu'une action d'un niveau antérieur (pas forcément le niveau précédent) établit ses préconditions et qu'une action qui suit requiert un de ses effets. L'existence de différents codages découle de la traduction des différentes stratégies de recherche utilisées dans les espaces de plans. Ces codages restent moins performants, en termes de compacité et de temps de résolution, que le meilleur codage dans les espaces d'états (celui avec frame-axiomes explicatifs) [49].

Dans tous ces codages, les ensembles d'actions indépendants qui peuvent faire partie d'un plan sont associés à des symboles d'étapes. Un ordre sur ces étapes détermine un ordre total pour l'exécution des ensembles d'actions indépendants. Deux étapes particulières sont créées : l'étape p_0 qui représente l'état initial du problème, et l'étape p_{k+1} qui représente le but du problème. Ces deux étapes ne contiendront aucune action. Elles sont équivalentes aux actions fictives *Start* et *End* qui servent à initialiser un plan partiel et sont généralement utilisées en planification dans les espaces de plans partiels. Pour prendre en compte ces deux étapes, l'ensemble F_a des fluents présents dans les ajouts des actions est augmenté des fluents de l'état initial I. De même, l'ensemble F_p des fluents présents dans les préconditions des actions est augmenté des fluents du but B. On peut produire des codages produisant moins de clauses en considérant les actions *Start* et *End* comme des cas particuliers [68], mais pour des raisons de compacité dans l'écriture des codages nous les présentons ici de manière intégrée aux autres actions.

Les règles des codages dans les espaces de plans produisent les propositions suivantes, pour une action a, un fluent f, et deux étapes p et q :

– $(a \in p)$ est vraie si et seulement si l'action a appartient à l'étape p.

- $\text{Adds}(p, f)$, est vraie si et seulement si l'étape p contient une action qui ajoute le fluent f.
- $\text{Needs}(p, f)$ est vraie si et seulement si l'étape p contient une action qui a f en précondition.
- $\text{Dels}(p, f)$ est vraie si et seulement si l'étape p contient une action qui retire le fluent f.
- $p \xrightarrow{f} q$ représente un lien causal et est vraie si et seulement si l'étape p produit le fluent f qui est une précondition de l'étape q.
- $p \prec q$ est vraie si et seulement si l'étape p précède l'étape q ; toutes les actions associées à p doivent donc être exécutées avant celles associées à q. Ces propositions traduisent un ordre partiel sur les étapes.

5.1 Partie commune des codages dans les espaces de plans

Elle permet d'établir une correspondance entre toutes les actions disponibles et celles qui font partie des étapes des plans solutions. Elle est constituée de cinq règles :

1. *État initial et but :* l'étape p_0 produit l'état initial du problème, et l'étape p_{k+1} requiert les buts.

$$\left[\bigwedge_{f \in I} \ \text{Adds}(p_0, f)\right] \wedge \left[\bigwedge_{f \in (F \setminus I)} \ \neg\text{Adds}(p_0, f)\right] \wedge \left[\bigwedge_{f \in B} \ \text{Needs}(p_{k+1}, f)\right]$$

2. *Correspondance action/étape :* si une étape ajoute, retire, ou a pour précondition un fluent, alors cette étape peut correspondre à n'importe quelle action qui a le même comportement vis-à-vis de ce fluent. Inversement, si une action appartient à une étape, ses préconditions, ajouts et retraits sont vrais.

$$\bigwedge_{i \in [1,k]} \ \bigwedge_{f \in F_a} \ \left[\text{Adds}(p_i, f) \Leftrightarrow \bigvee_{a \in A \mid f \in \text{Add}(a)} \ (a \in p_i)\right]$$

$$\bigwedge_{i \in [1,k]} \ \bigwedge_{f \in F_d} \ \left[\text{Dels}(p_i, f) \Leftrightarrow \bigvee_{a \in A \mid f \in \text{Del}(a)} \ (a \in p_i)\right]$$

$$\bigwedge_{i \in [1,k]} \ \bigwedge_{f \in F_p} \ \left[\text{Needs}(p_i, f) \Leftrightarrow \bigvee_{a \in A \mid f \in \text{Prec}(a)} \ (a \in p_i)\right]$$

3. *Exclusions mutuelles :* deux actions qui ne sont pas indépendantes ne peuvent appartenir à une même étape.

$$\bigwedge_{i \in [1,k]} \ \bigwedge_{\{a_m, a_n\} \subseteq A \mid m < n \wedge \neg(a_m \| a_n)} \ [\neg(a_m \in p_i) \vee \neg(a_n \in p_i)]$$

5.2 Codage par liens causaux, protection d'intervalles et ordre partiel

La production des préconditions se fait par le codage direct des liens causaux. La proposition $p \xrightarrow{f} q$ traduit le fait que l'étape p produit le fluent f, précondition de l'étape q. La protection de f dans l'intervalle compris entre les deux étapes se fait par promotion (l'étape menaçant f est placée avant p), ou par rétrogradation (l'étape menaçant f est placée après q). L'ordre d'exécution des étapes est donné par une relation de précédence de la forme $p \prec q$. Les quatre règles suivantes sont ajoutées à la partie commune :

1. *Production des préconditions :* chaque fluent est supporté par un lien causal entre l'étape qui le produit et l'étape qui l'utilise en tant que précondition.

$$\bigwedge_{i \in [1,k+1]} \ \bigwedge_{f \in F_p} \ \left[\text{Needs}(p_i, f) \Rightarrow \left(\ f \in F_a \hookrightarrow \bigvee_{j \in [0,k] \mid j \neq i} \ p_j \xrightarrow{f} p_i \ \middle| \ \bot \ \right)\right]$$

2. *Liens causaux :* un lien causal supportant un fluent f entre deux étape p_i et p_j implique que p_i ajoute f, que p_j a pour précondition f, et que p_i précède p_j.

$$\bigwedge_{i\in[0,k]}\ \bigwedge_{j\in[1,k+1]\,|\,i\neq j}\ \bigwedge_{f\in(F_p\cap F_a)}\ \left[p_i \xrightarrow{f} p_j \Rightarrow (\mathrm{Adds}(p_i,f)\wedge \mathrm{Needs}(p_j,f)\wedge p_i \prec p_j)\right]$$

3. *Protection d'intervalles :* si un lien causal supporte le fluent f entre une étape p_i et une étape p_j, et si une étape p_q retire f, alors p_q doit précéder p_i (promotion) ou p_j doit précéder p_q (rétrogradation).

$$\bigwedge_{i\in[0,k]}\ \bigwedge_{j\in[1,k+1]\,|\,i\neq j}\ \bigwedge_{q\in[1,k]\,|\,q\neq i\wedge q\neq j}\ \bigwedge_{f\in(F_p\cap F_a\cap F_d)}\ \left[\begin{array}{l} p_i \xrightarrow{f} p_j \wedge \mathrm{Dels}(p_q,f) \Rightarrow \\ (p_q \prec p_i \vee p_j \prec p_q) \end{array}\right]$$

4. *Propriétés de la relation de précédence :* transitive et antisymétrique. L'irréflexivité est codée dans les autres règles par le biais des indices des étapes.

$$\bigwedge_{i\in[0,k+1]}\ \bigwedge_{j\in[0,k+1]\,|\,i\neq j}\ \bigwedge_{q\in[0,k+1]\,|\,q\neq i\wedge q\neq j}\ ((p_i \prec p_j \wedge p_j \prec p_q) \Rightarrow p_i \prec p_q)$$

$$\bigwedge_{i\in[0,k]}\ \bigwedge_{j\in[i+1,k+1]\,|\,i\neq j}\ \neg(p_i \prec p_j) \vee \neg(p_j \prec p_i)$$

5.3 Codage par liens causaux, protection d'intervalles et étapes contiguës

Ce codage simplifie le précédent dans lequel plusieurs modèles correspondent au même plan solution, à la numérotation des étapes près. Les étapes doivent maintenant suivre un ordre prédéfini d'indice croissant, ce qui rend inutile la relation de précédence. La protection d'intervalles ne nécessite plus la promotion ou la rétrogradation avec un ordre partiel sur les étapes, mais se réalise en interdisant qu'une étape comprise dans un intervalle défini par un lien causal ne le menace. Les deux règles suivantes sont ajoutées à la partie commune :

1. *Production des préconditions :* chaque fluent est supporté par un lien causal entre l'étape qui le produit et l'étape qui l'utilise en tant que précondition.

$$\bigwedge_{i\in[1,k]}\ \bigwedge_{f\in F_p}\ \left[\mathrm{Needs}(p_i,f) \Rightarrow \left(\ f\in F_a \hookrightarrow \bigvee_{j\in[1,i-1]}\ p_j \xrightarrow{f} p_i\ \middle|\ \bot\ \right)\right]$$

2. *Liens causaux et protection d'intervalles :* la présence d'un lien causal supportant un fluent f entre deux étapes p_i et p_j implique que p_i ajoute f et que p_j a pour précondition f. De plus, une étape qui retire f ne peut s'insérer entre p_i et p_j.

$$\bigwedge_{i\in[1,k-1]}\ \bigwedge_{j\in[i+1,k]}\ \bigwedge_{f\in(F_p\cap F_a)}\ \left[\begin{array}{l} p_i \xrightarrow{f} p_j \Rightarrow \mathrm{Adds}(p_i,f)\wedge \mathrm{Needs}(p_j,f)\wedge \\ \left(\ f\in F_d \hookrightarrow \bigwedge_{q\in[i+1,j-1]}\ \neg\mathrm{Dels}(p_q,f)\ \middle|\ \top\ \right) \end{array}\right]$$

5.4 Codage du « chevalier blanc »

Ce codage est actuellement le plus compact des codages dans les espaces de plans en nombre de variables et de clauses produites ; il est également le plus performant par son temps de résolution. Il exprime directement les liens causaux : si une étape requiert un fluent, c'est qu'une étape précédente doit l'avoir créé. On n'utilise ainsi que les variables déjà présentes dans la partie commune. La protection d'intervalles se réalise en codant la technique du « chevalier blanc »introduite par le planificateur TWEAK [16]. Les deux règles suivantes sont ajoutées à la partie commune :

1. *Production des préconditions :* la précondition d'une étape d'un niveau i doit être ajoutée par une étape précédente.

$$\bigwedge_{i\in[1,k+1]}\ \bigwedge_{f\in F_p}\ \Big[\text{Needs}(p_i,f) \Rightarrow \Big(\ f \in F_a \hookrightarrow \bigvee_{j\in[0,i-1]}\ \text{Adds}(p_j,f)\ \Big|\ \bot\ \Big)\Big]$$

2. *Chevalier blanc :* si une étape a pour précondition le fluent f au niveau i, et qu'une autre le retire au niveau j avant que la première puisse l'utiliser, alors il doit y avoir une troisième étape qui rétablit f à un niveau q tel que $j < q < i$.

$$\bigwedge_{i\in[3,k+1]}\ \bigwedge_{j\in[1,i-2]}\ \bigwedge_{f\in(F_p\cap F_d)}\ \left[\begin{array}{l}\text{Needs}(p_i,f)\wedge\text{Dels}(p_j,f)\Rightarrow\\ \Big(\ f\in F_a \hookrightarrow \bigvee_{q\in[j+1,i-1]}\ \text{Adds}(p_q,f)\ \Big|\ \bot\ \Big)\end{array}\right]$$

6 Codages du graphe de planification

Ces codages sont basés sur le graphe de planification utilisé initialement dans le planificateur GRAPHPLAN [7, 8]. Ils reprennent les mêmes principes que les codages dans les espaces d'états, tout en tirant parti des informations fournies par le graphe de planification (niveau d'apparition des actions dans le graphe, exclusions mutuelles propagées pendant la construction). Nous ne donnerons pas de description formelle du graphe de planification. On peut se reporter à [68, 59] pour une présentation plus détaillée. Nous allons toutefois présenter brièvement la construction d'un graphe de planification au travers d'un exemple, en explicitant les notations utilisées pour les codages.

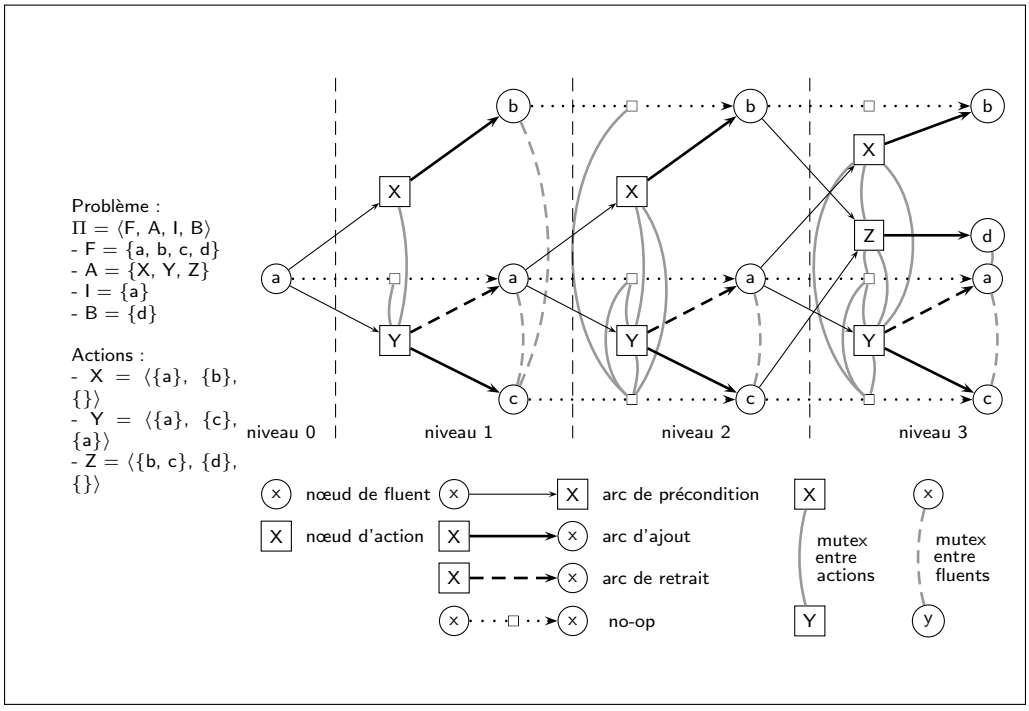

FIGURE 1 – Le graphe de planification pour un problème simple

La figure 1 représente le graphe de planification construit jusqu'au niveau $k = 3$ pour un problème de planification Π contenant 4 fluents et 3 actions. Le seul plan solution que l'on peut extraire de ce graphe est $P = \langle\{A\},\{B\},\{C\}\rangle$. Comme pour le codage dans

les espaces d'états avec no-ops, on considère que l'ensemble des actions A contient aussi un no-op pour chaque fluent possible. Une action ou un fluent apparaissant plusieurs fois dans le graphe correspond à différents nœuds, un pour chaque niveau d'apparition. Cependant, pour alléger la description de l'exemple, nous parlerons simplement, au lieu de nœuds d'actions et de fluents, des actions et des fluents qui leurs sont associés.

Le niveau 0 contient l'unique fluent a présent dans l'état initial. Au niveau 1, les actions ayant toutes leurs préconditions présentes au niveau 0 (X et Y) sont introduites dans le graphe. De plus, un no-op est ajouté pour chaque fluent du niveau 0. Les effets des actions sont ensuite introduits dans le graphe, et des arcs représentant les liens entre une action et ses préconditions et effets sont créés. Des arcs représentant les contraintes d'exclusion mutuelle (mutex) sont ensuite ajoutés. Ainsi, comme Y retire a, Y et le no-op de a sont mutex (puisque a est à la fois une précondition et un ajout du no-op) ainsi que Y et X (puisque a est une précondition de X). Ce mutex représente le fait qu'il ne peut y avoir de plan parallèle où X et Y seraient tous les deux présents au niveau 1. Comme les actions permettant de produire b (seulement X) et c (seulement Y) sont toutes mutex deux à deux, b et c sont mutex au niveau 1. De même, a et c sont mutex. Ce processus et ensuite répété pour le passage du niveau 1 au niveau 2, en enregistrant en plus comme mutex les paires d'actions pour lesquelles un couple de fluents formé par une précondition de l'une et de l'autre est mutex. Ainsi, le no-op de b et le no-op de c sont mutex au niveau 2. De plus, une action ne peut être introduite dans le graphe si deux de ses préconditions sont mutex. Ainsi, Z ne peut être introduite au niveau 2 puisque b et c sont mutex. C'est là un point clef : l'apparition des actions est des fluents est retardée grâce à la propagation des exclusions mutuelles. Les fluents b et c ne sont plus mutex au niveau 2, puisqu'il existe un couple d'actions non mutex à ce niveau permettant de les produire : le no-op de b et l'action Y. Ainsi, Z peut être introduite au niveau 3, produisant le fluent du but d. Le codage sous forme de base de clauses peut être effectué ; en cas d'échec, le graphe sera étendu d'un niveau supplémentaire et le processus relancé.

Chaque nœud du graphe peut être associé à une unique proposition, qui seront les seules propositions utilisées pour les codages. On notera, pour un graphe de planification GP :

- NoeudFluent(GP) : l'ensemble des nœuds correspondant à des fluents.
- NoeudInit(GP) : l'ensemble des nœuds du niveau 0 associés aux fluents de l'état initial.
- NoeudBut(GP) : l'ensemble des nœuds du niveau k associés aux fluents du but.
- ArcsPrec(GP) l'ensemble des arcs (n_f, n_a) où f est une précondition de a.
- ArcsAdd(GP) l'ensemble des arcs (n_a, n_f) où f est un ajout de a.
- MutexActions(GP) l'ensemble des arcs (n_a, n_b) où a et b sont mutex au niveau correspondant.

6.1 Codage du graphe avec actions, fluents et mutex

Ce codage comporte une proposition pour chacun des nœuds du graphe, correspondant aux actions et aux fluents. Il comporte les quatre règles suivantes :

1. *État initial et but :* les fluents de l'état initial et du but sont vrais.

$$\left[\bigwedge_{n_f \in \text{NoeudsInit}(GP)} n_f\right] \wedge \left[\bigwedge_{n_f \in \text{NoeudsBut}(GP)} n_f\right]$$

2. *Préconditions des actions :* si une action est utilisée, ses préconditions sont vraies au niveau précédent.

$$\bigwedge_{(n_f, n_a) \in \text{ArcsPrec}(GP)} [n_a \Rightarrow n_f]$$

3. *Production des fluents :* si un fluent est vrai à un niveau supérieur à l'état initial, la disjonction des actions qui peuvent le produire au niveau précédent est vraie (no-ops compris s'il est présent à ce niveau). Ces clauses correspondent aux arcs d'ajout calculés à la construction du graphe.

$$\bigwedge_{n_f \in (\text{NoeudsFluent}(GP) \setminus \text{NoeudsInit}(GP))} \left[n_f \Rightarrow \bigvee_{(n_a, n_f) \in \text{ArcsAdd}(GP)} n_a \right]$$

4. *Exclusions mutuelles :* les actions mutuellement exclusives ne peuvent pas être vraies en même temps. Ces mutex concernent l'indépendance entre actions et les mutex d'atteignabilité trouvés lors de la construction du graphe.

$$\bigwedge_{\{n_a, n_b\} \in \text{MutexActions}(GP)} [\neg n_a \vee \neg n_b]$$

6.2 Codage du graphe avec actions et mutex

Ce codage ne fait plus intervenir les nœuds de fluent. Le nombre de propositions est réduit mais le nombre de clauses augmente. Il est composé des trois règles suivantes :

1. *But :* pour chacun des fluents du but, au moins une action qui le produit doit avoir été appliquée.

$$\bigwedge_{n_f \in \text{NoeudsBut}(GP)} \left[\bigvee_{(n_a, n_f) \in \text{ArcsAdd}(GP)} n_a \right]$$

2. *Production des préconditions :* les préconditions de chaque action du plan, si elles ne sont pas dans l'état initial, doivent être établies par au moins une action.

$$\bigwedge_{(n_f, n_a) \in \text{ArcsPrec}(GP) \mid n_f \notin \text{NoeudsInit}(GP)} \left[n_a \Rightarrow \bigvee_{(n_b, n_f) \in \text{ArcsAdd}(GP)} n_b \right]$$

3. *Exclusions mutuelles :* règle 4 du codage précédent.

$$\bigwedge_{\{n_a, n_b\} \in \text{MutexActions}(GP)} [\neg n_a \vee \neg n_b]$$

7 Notes bibliographiques

Nous présentons dans cette section les principaux travaux ayant trait à la planification SAT. Pour conserver une certaine cohérence à ces notes, nous avons préféré suivre, dans la mesure du possible, un ordre chronologique. Pour avoir accès à des études plus détaillées en français sur la planification de type SAT, on se reportera à [68, 50, 59].

L'article original de SATPLAN [37] montre comment encoder manuellement des problèmes de planification du domaine des cubes sous forme de problèmes SAT. Les codages « linéaires »utilisés sont issus du calcul des situations [53] et nécessitent l'utilisation de « frame-axiomes »qui indiquent, pour chaque action, que les fluents qu'elle n'affecte pas conservent leur valeur de vérité après son application. Le codage utilisé impose l'application séquentielle des actions ce qui entraîne une augmentation importante de la taille du codage (en nombre de variables, de clauses) avec le nombre d'opérateurs des plans. [38] montrent comment encoder le graphe de planification de GRAPHPLAN [7, 8] comme un problème SAT pour en extraire un plan solution. Ils développent les codages dans les espaces d'états avec no-ops et avec frame-axiomes explicatifs. Ils remarquent que l'utilisation de ces derniers permet le parallélisme, contrairement aux frame-axiomes classiques qui imposent l'utilisation d'une unique action à chaque niveau. Ils comparent leur système SATPLAN aux planificateurs de référence UCPOP [57] et GRAPHPLAN et montrent que l'approche SAT

peut être compétitive. Les codages de [38] seront largement repris, étudiés et améliorés, en particulier dans [23, 49, 68, 50].

[36] montrent comment encoder automatiquement des problèmes de planification en problèmes SAT. Ils introduisent l'utilisation du parallélisme et proposent la première version des codages dans les espaces de plans partiels. Ils comparent la taille de différents codages.

[23] étudient l'impact de différentes représentations des actions dans les codages des problèmes de planification. Leur système MEDIC est le premier qui prend directement en entrée la description STRIPS d'un problème de planification pour le résoudre automatiquement grâce à un prouveur SAT. [4], dans leur système CSATPLAN, encodent le graphe de planification de GRAPHPLAN pour produire une base de clause pour laquelle des prouveurs SAT cherchent ensuite un modèle. [48] améliorent les performances des codages dans les espaces de plans en réduisant le nombre de variables et de clauses nécessaires. [47] étudient l'impact de plusieurs opérations de raffinement (ordre total / partiel, recherche avant / arrière / bidirectionnelle, espaces d'états / de plans) sur la taille et l'efficacité de différents codages. [27] remarquent que les valeurs de vérités des fluents sont induites par celles des actions. Ils modifient le prouveur TABLEAU [19] pour que les choix effectués dans la procédure de Davis et Putnam ne portent que sur les actions. Les valeurs des fluents sont alors déduites par une simple propagation unitaire. Les performances du prouveur SAT ainsi obtenu sont comparées avec sa version originale sur plusieurs des codages de [23]. [49] réalisent une comparaison systématique entre les codages dans les espaces d'états et de plans. Ils améliorent les notations, proposent deux codages plus efficaces que ceux de [38] et [23], et démontrent également que les codages les plus compacts sont ceux des espaces d'états.

[39, 41] développent le planificateur BLACKBOX qui encode les actions, fluents, et exclusions mutuelles du graphe de planification de GRAPHPLAN comme un problème SAT pour en extraire une solution. Ce planificateur se montre plus performant que SATPLAN et GRAPHPLAN. BLACKBOX permet l'utilisation de la procédure d'extraction de GRAPHPLAN ainsi que celle de plusieurs prouveurs SAT. Il possède de nombreuses options et permet l'exécution séquentielle de ces prouveurs sur une certaine durée, avec leurs différentes options de fonctionnement.

[46, 44] montrent comment encoder la planification de type HTN (Hierarchical Task Network), et donnent la taille des codages ainsi obtenus. [45] développe plusieurs heuristiques qui, grâce aux HTN, utilisent des connaissances sur le domaine pour générer des codages souvent plus compacts et plus faciles à résoudre. [40] montrent comment améliorer les performances de SATPLAN en codant des connaissances spécifiques au domaine de manière déclarative et indépendante du prouveur utilisé. Le planificateur TLPLAN [1, 2], efficace sur de nombreux domaines, utilise un langage de représentation des connaissances sous forme déclarative, expressif et basé sur une logique temporelle du premier ordre. Il effectue une recherche en chaînage avant dans les espaces d'états, en vérifiant, à chaque état, la validité de formules de ce langage qui représentent les connaissances sur le domaine. [40] montrent que ce type de contrôle d'information est possible dans les approches SAT et améliore beaucoup les performances. [31] montrent que les connaissances que l'on peut représenter dans le langage de TLPLAN peuvent être classées en trois catégories : (1) des connaissances statiques basées sur l'état initial et le but qui servent à éliminer directement des actions avant même de les utiliser dans la base de clauses, (2) des connaissances qui dépendent de l'état courant et peuvent être exprimées par ajout de clauses dans la base sans créer de nouvelles variables, (3) des connaissances qui dépendent de l'état courant et qui nécessitent la création de nouvelles variables. Ils encodent les deux premières catégories dans BLACKBOX pour plusieurs domaines, et apportent une réduction significative du temps de recherche. La troisième catégorie semble être inutilisable, car elle augmente la taille de la base de clause de manière trop importante.

Pour traiter plus spécifiquement des problèmes de planification, [67] implémentent le prouveur MODOC. Il est basé sur des techniques de résolution en chaînage arrière qui lui permettent de se focaliser sur les clauses qui correspondent aux buts du problème. Il est également capable de déterminer les clauses qui ne sont pas pertinentes pour la résolution du problème par rapport à une interprétation partielle, ce qui lui permet de compléter plus efficacement cette interprétation partielle pour chercher un modèle. Les performances de MODOC sont compétitives avec celles du prouveur incomplet WALKSAT [66], mais sont beaucoup moins bonnes que celles du prouveur complet SATO [78] qui possède une implémentation extrêmement efficace de la propagation unitaire.

Lorsque l'on crée une base de clauses à partir d'un problème de planification, on perd la structure de ce dernier alors qu'elle peut servir à guider la résolution. Pour pallier cet inconvénient, [60] propose un algorithme qui évite cette transformation, en appliquant une méthode inspirée de la procédure de Davis et Putnam [21, 20] directement sur les actions et les fluents que l'on peut obtenir à partir de la description STRIPS du problème (pour un nombre de niveaux donné). Cet algorithme utilise des règles de propagation sur des clauses issues d'un codage parallèle dans les espaces d'états du problème avec frame-axiomes explicatifs. D'autres règles de propagation sont appliquées, correspondant aux effets de la propagation unitaire dans un prouveur SAT. Les performances observées sont largement supérieures à celles de GRAPHPLAN, mais inférieures à celles de SATZ et WALKSAT. Cette idée est reprise dans le planificateur DPPLAN [5], qui utilise une procédure de Davis et Putnam pour extraire un plan solution directement du graphe de planification. Plusieurs fonctions propagent des valeurs attachées aux nœuds d'actions et de fluents, ce qui donne une procédure de recherche bidirectionnelle. Les performances de DPPLAN sont comparables à celles obtenues avec SATZ. Cette approche est perfectionnée dans le planificateur LCDPP [68, 69] qui utilise la relation d'autorisation [14, 15] et améliore ainsi les performances de DPPLAN.

[22] proposent une méthode qui permet de déterminer, à partir du graphe de planification, la pertinence des fluents pour le but du problème. Elle calcule des exclusions mutuelles binaires entre fluents et actions correspondant à leur pertinence respective : en employant une des deux actions, on exclut l'autre, et le prouveur SAT peut ainsi effectuer des propagations. Les performances obtenues avec le prouveur RELSAT sont améliorées de façon modérée (au plus 6 fois), alors qu'elles sont dégradées avec le prouveur SATZ.

[68, 50] simplifient les codages originaux (clauses redondantes), introduisent le parallélisme dans les codages d'espaces de plans, codent le graphe de planification par ses seules actions et exclusions mutuelles, utilisent la relation d'autorisation et produisent des bases de clauses plus compactes (en nombre de clauses et de variables). Ils implémentent le système TSP (Tunable SatPlan) et réalisent de nombreux tests comparatifs.

[61] montre comment, pour des problèmes de planification présentant des symétries, on peut rajouter aux codages des contraintes de rupture de symétrie qui augmentent de manière importante les performances de la résolution.

Les premières approches de la planification SAT [37, 38, 36] s'intéressent à des codages pour lesquels on connaît à l'avance la longueur minimale d'un plan solution. Dans les approches suivantes [23, 4, 41], la stratégie de résolution standard cherche à satisfaire des codages représentant tous les plans d'une longueur fixée en commençant à une longueur minimale (soit 1, soit une longueur obtenue grâce à la construction d'un graphe de planification permettant d'obtenir les fluents du but). Tant qu'un plan solution n'est pas trouvé par la résolution de ce codage, cette longueur est augmentée d'une unité pour produire le codage suivant, ce qui garantit la production d'un plan solution de nombre d'étapes minimal. [62, 65] étudient des stratégies basées sur l'évaluation parallèle ou intercalée de plusieurs formules. Ces stratégies peuvent éviter l'évaluation de certaines formules inconsistantes très difficiles et permettent des améliorations importantes des temps de résolution au prix d'une perte de la garantie d'optimalité du plan solution en nombre de

niveaux.

Alors que l'approche classique considère que l'application parallèle d'opérateurs n'est possible que s'ils sont indépendants, [50, 64] montrent comment on peut améliorer les performances de la résolution en employant la relation d'autorisation à la place de celle d'indépendance.

Le planificateur SATPLAN'04 [34], qui est une mise à jour du planificateur BLACKBOX, remporte la compétition IPC-2004 dans la catégorie des planificateurs optimaux (en nombre de niveaux du plan). Ce résultat est inattendu puisque, par rapport à la version précédente de BLACKBOX, SATPLAN'04 n'intègre pas de nouveau codage ou prétraitement, et pas non plus de procédure de propagation des mutex. [35] analyse les raisons de ce succès au premier plan desquelles il place les gains en performance des prouveurs SAT, la difficulté des problèmes posés et leur forte structuration.

Le système MAXPLAN [18, 76, 79] remporte, à égalité avec SATPLAN'06 [42], la compétition IPC-2006 dans la catégorie des planificateurs optimaux en intégrant à SATPLAN'04 des mutex de longues distances (londex). Ceux-ci permettent une réduction de l'espace de recherche : ils généralisent les mutex classiques et peuvent capturer des contraintes entre actions situées à différents niveaux. Ils sont automatiquement générés par une analyse d'un graphe de transition du domaine de planification, lui-même construit à partir d'une représentation multivaluée du domaine. SATPLAN'06 intègre une procédure de propagation des mutex mais seulement pour les fluents (la prise en compte des mutex d'actions entraîne une saturation rapide de la mémoire).

Plusieurs études cherchent maintenant à étendre l'utilisation des techniques SAT pour sortir du cadre « classique »de la planification (environnement statique, observabilité totale, agent omniscient, actions atomiques et déterministes). [74, 75] proposent ainsi, dans leur planificateur LPSAT, une extension de la planification SAT pour la résolution de problèmes de planification avec prise en compte de ressources. Le principe consiste à résoudre un problème constitué par une base de clauses et un ensemble de contraintes sous forme d'équations linéaires et d'inégalités, ces contraintes étant déclenchées par l'assignation de la valeur de vérité vrai à certaines variables de la base. On peut par exemple associer à une variable propositionnelle une contrainte précisant que la quantité de carburant disponible pour un véhicule doit être supérieure à une certaine valeur. Lorsque la valeur de cette variable passe à vrai, il faut s'assurer que cette contrainte est vérifiée, ainsi que les autres contraintes qui dépendent de cette variable. Dans le cas contraire, un retour arrière dans la procédure de Davis et Putnam est effectué, même si la base de clauses reste consistante. Le planificateur LPSAT utilise à cet effet une version modifiée de RELSAT [6], et le prouveur de contraintes arithmétiques linéaires CASSOWARY [10]. [29] développent une méthode permettant d'encoder des problèmes de planification numérique sous forme de problèmes SAT. Cette approche utilise une approximation de l'atteignabilité des variables du domaine. Ils la comparent à une autre traduction, plus directe utilisant « SAT Modulo Theory », extension de SAT avec des variables numériques et des contraintes arithmétiques. Dans les exemples qu'ils ont pu pratiquement tester, leur approche, optimale en nombre de niveaux, est plus efficace que la traduction avec SMT et rivalise même avec des planificateurs non optimaux. [51] étendent la planification SAT aux buts temporellement étendus (TEG : Temporally Extended Goals) qui permettent d'exprimer des contraintes qui imposent le passage par certains états pendant l'exécution du plan solution. Plusieurs travaux [12, 13, 24] s'intéressent également à l'utilisation des techniques SAT pour la planification dans l'incertain dans le cas bien particulier de l'observabilité nulle (dans ce cas, on parle de « conformant planning »). [63] propose également plusieurs codages QBF (Quantified Boolean Formulae : extension de SAT avec des quantificateurs) pour ce type de problèmes de planification.

8 Conclusion

Nous avons présenté dans ce chapitre une formalisation des principaux codages de problèmes de planification sous forme de bases de clauses. L'objectif de cette transformation est de bénéficier automatiquement des progrès constants effectués dans la conception des prouveurs SAT. Les codages que nous avons présentés, souvent décrits de façon informelle dans la littérature, sont inspirés des principales techniques de planification : planification dans les espaces d'états et de plans, et planification par recherche dans les graphes de planification. Les planificateurs tels SATPLAN'06 ou MAXPLAN utilisant ce principe sont actuellement les plus efficaces pour la recherche de plans parallèles optimaux, comme en témoignent les résultats des compétitions internationales de planification où ils se retrouvent systématiquement en tête (voir le site de la cinquième compétition internationale de planification IPC-2006 `http://zeus.ing.unibg.it/ipc-5`). Néanmoins, l'intérêt pratique de calculer des plans parallèles n'est pas évident : d'abord l'optimalité ne concerne que le nombre de niveaux des plans et elle n'est pas corrélée à l'optimalité en nombre d'actions ; ensuite, il s'agit d'une restriction de la planification temporelle où toutes les actions ont une durée uniforme, et l'extension de ces techniques dans un cadre temporel plus général semble difficilement possible (du moins en utilisant des prouveurs SAT standards). En effet, on ne peut raisonnablement pas envisager de discrétiser le temps dans le contexte de la planification : le nombre de variables nécessaires pour représenter l'application d'une action pour chaque instant s'avère en pratique bien trop important.

Références

[1] F. Bacchus and F. Kabanza. Planning for temporally extended goals. In *Proc. AAAI-96*, pages 1215–1222, 1996.

[2] F. Bacchus and F. Kabanza. Using temporal logics to express search control knowledge for planning. *Artificial Intelligence*, 116(1-2) :123–191, 2000.

[3] C. Bäckström. Computationnal aspects of reordering plans. *JAIR*, 9 :99–137, 1998.

[4] M. Baioletti, S. Marcugini, and A. Milani. An extension of SATPLAN for planning with constraints. In *Proc. AIMSA-98*, pages 39–49, 1998.

[5] M. Baioletti, S. Marcugini, and A. Milani. DPPlan : An algorithm for fast solutions extraction from a planning graph. In *Proc. AIPS-00*, pages 13–21, 2000.

[6] R. J. Bayardo and R. Schrag. Using CSP look-back techniques to solve real-world SAT instances. In *Proc. AAAI-97*, pages 203–208, 1997.

[7] A. Blum and M. Furst. Fast planning through planning-graphs analysis. In *Proc. IJCAI-95*, pages 1636–1642, 1995.

[8] A. Blum and M. Furst. Fast planning through planning-graphs analysis. *Artificial Intelligence*, 90(1-2) :281–300, 1997.

[9] B. Bonet and H. Geffner. Planning as heuristic search : New results. In *Proc. ECP-99*, pages 360–372, 1999.

[10] A. Borning, K. Marriott, P. J. Stuckey, and Y. Xiao. Solving linear arithmetic constraints for user interface applications. In *Proc. ACM Symposium on User Interface Software and Technology*, pages 87–96, 1997.

[11] T. Bylander. The computational complexity of STRIPS planning. *Artificial Intelligence*, 69(1-2) :165–204, 1994.

[12] C. Castellini, E. Giunchiglia, and A. Tacchella. Improvements to SAT-based conformant planning. In *Proc. ECP-01*, pages 241–252, 2001.

[13] C. Castellini, E. Giunchiglia, and A. Tacchella. SAT-based planning in complex domains : Concurrency, constraints and nondeterminism. *Artificial Intelligence*, 147(1-2) :85–118, 2003.

[14] M. Cayrol, P. Régnier, and V. Vidal. New results about LCGP, a least committed Graphplan. In *Proc. AIPS-00*, pages 273–282, 2000.

[15] M. Cayrol, P. Régnier, and V. Vidal. Least commitment in Graphplan. *Artificial Intelligence*, 130(1) :85–118, 2001.

[16] D. Chapman. Planning for conjunctive goals. *Artificial Intelligence*, 32(3) :333–377, 1987.

[17] Y. Chen, C. Hsu, and B. Wah. Partitioning and resolution in SGPlan. *JAIR*, 26 :323–369, 2006.

[18] Y. Chen, X. Zhao, and W. Zhang. Long distance mutual exclusion for propositional planning. In *Proc. IJCAI-07*, pages 1840–1845, 2007.

[19] J. M. Crawford and L. D. Auton. Experimental results on the crossover point in random 3-SAT. *Artificial Intelligence*, 81(1-2) :31–57, 1996.

[20] M. Davis, G. Logemann, and D. Loveland. A computing procedure for quantification theory. *Communications of the ACM*, 5 :394–397, 1962.

[21] M. Davis and H. Putnam. A computing procedure for quantification theory. *Journal of the ACM*, 7 :201–215, 1960.

[22] M. B. Do, B. Srivastava, and S. Kambhampati. Investigating the effect of relevance and reachability constraints in SAT encodings of planning. In *Proc. AIPS-00*, pages 308–314, 2000.

[23] M. Ernst, T. Millstein, and D.S. Weld. Automatic SAT-compilation of planning problems. In *Proc. IJCAI-97*, pages 1169–1177, 1997.

[24] P. Ferraris and E. Giunchiglia. Planning as satisfiability in nondeterministic domains. In *Proc. AAAI-00*, pages 748–753, 2000.

[25] R.E. Fikes and N.J. Nilsson. STRIPS : A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2(3-4) :189–208, 1971.

[26] M. Fox and D. Long. The automatic inference of state invariants in TIM. *JAIR*, 9 :367–421, 1998.

[27] E. Giunchiglia, A. Massarotto, and R. Sebastiani. Act, and the rest will follow : Exploiting determinism in planning as satisfiability. In *Proc. AAAI-98*, pages 948–953, 1998.

[28] M. Helmert. A planning heuristic based on causal graph analysis. In *Proc. ICAPS-04*, pages 161–170, 2004.

[29] J. Hoffmann, H. Kautz, C. Gomes, and B. Selman. SAT encodings of state-space reachability problems in numeric domains. In *Proc. IJCAI-07*, pages 1918–1923, 2007.

[30] J. Hoffmann and B. Nebel. The FF planning system : Fast plan generation through heuristic search. *JAIR*, 14 :253–302, 2001.

[31] Y.C. Huang, B. Selman, and H. Kautz. Control knowledge in planning : Benefits and tradeoffs. In *Proc. AAAI-99*, pages 511–517, 1999.

[32] S. Kambhampati. Planning graph as a (dynamic) CSP : Exploiting EBL, DDB and other CSP techniques in Graphplan. *JAIR*, 12 :1–34, 2000.

[33] S. Kambhampati and B. Srivastava. Universal Classical Planner : An algorithm for unifying state-space and plan-space planning. In *Proc. EWSP-95*, pages 93–94, 1995.

[34] H. Kautz. SATPLAN'04 : Planning as satisfiability. In *Abstracts of the 4th International Planning Competition (IPC-04)*, 2004.

[35] H. Kautz. Deconstructing planning as satisfiability. In *Proc. AAAI-06*, 2006.

[36] H. Kautz, D. McAllester, and B. Selman. Encoding plans in propositional logic. In *Proc. KR-96*, pages 374–384, 1996.

[37] H. Kautz and B. Selman. Planning as satisfiability. In *Proc. ECAI-92*, pages 359–363, 1992.

[38] H. Kautz and B. Selman. Pushing the envelope : Planning, propositional logic and stochastic search. In *Proc. AAAI-96*, pages 1194–1201, 1996.

[39] H. Kautz and B. Selman. BLACKBOX : A new approach to the application of theorem proving to problem solving. In *Proc. AIPS-98 Workshop on Planning as Combinatorial Search*, 1998.

[40] H. Kautz and B. Selman. The role of domain-specific knowledge in the planning as satisfiability framework. In *Proc. AIPS-98*, pages 181–189, 1998.

[41] H. Kautz and B. Selman. Unifying SAT-based and Graph-based planning. In *Proc. IJCAI-99*, pages 318–325, 1999.

[42] H. Kautz, B. Selman, and J. Hoffmann. SATPLAN'06 : Planning as satisfiability. In *Abstracts of the 5th International Planning Competition (IPC-06)*, 2006.

[43] J. Koehler, B. Nebel, J. Hoffmann, and Y. Dimopoulos. Extending planning-graphs to an ADL subset. In *Proc. ECP-97*, pages 273–285, 1997.

[44] A. Mali. Hierarchical task network planning as satisfiability. In *Proc. ECP-99*, pages 122–134, 1999.

[45] A. Mali. Enhancing htn planning as satisfiability. In *Proc. ASC-00*, pages 325–333, 2000.

[46] A. Mali and S. Kambhampati. Encoding HTN planning in propositional logic. In *Proc. AIPS-98*, pages 190–198, 1998.

[47] A. Mali and S. Kambhampati. Frugal propositional encodings for planning. In *Proc. AIPS-98 Workshop planning as combinatorial search*, 1998.

[48] A. Mali and S. Kambhampati. Refinement-based planning as satisfiability. In *Proc. AIPS-98 Workshop planning as combinatorial search*, 1998.

[49] A. Mali and S. Kambhampati. On the utility of plan-space (causal) encodings. In *Proc. AAAI-99*, pages 557–563, 1999.

[50] F. Maris, P. Régnier, and V. Vidal. Planification sat : amélioration des codages et traduction automatique. In *Actes RFIA-04*, pages 1019–1028, Toulouse, France, 2004.

[51] R. Mattmüller and J. Rintanen. Planning for temporally extended goals as propositional satisfiability. In *Proc. IJCAI-07*, pages 1966–1971, 2007.

[52] J. McCarthy and P. J. Hayes. Some philosophical problems from the standpoint of artificial intelligence. *Machine Intelligence*, 4 :463–502, 1969.

[53] J. McCarthy and P.J. Hayes. Some philosophical problems from the standpoint of AI. *Machine Intelligence*, 4 :463–502, 1969.

[54] D. McDermott, M. Ghallab, A. Howe, C. Knoblock, A. Ram, M. Veloso, and D. Weld. *The Planning Domain Definition Language*. 1998.

[55] X.L. Nguyen and S. Kambhampati. Reviving partial order planning. In *Proc. IJCAI-01*, pages 459–466, 2001.

[56] X.L. Nguyen, S. Kambhampati, and R.S. Nigenda. Planning graph as the basis for deriving heuristics for plan synthesis by state space and CSP search. *Artificial Intelligence*, 135(1-2) :73–123, 2002.

[57] J. S. Penberthy and D. Weld. Ucpop : A sound, complete, partial order planner for adl. In *Proc. of KR-1992*, pages 103–114, 1992.

[58] P. Régnier and B. Fade. Complete determination of parallel actions and temporal optimization in linear plans of actions. In *Proc. EWSP-91*, pages 100–111, 1991.

[59] P. Régnier and V. Vidal. *Algorithmique de la planification en IA*. Cépaduès, 2004.

[60] J. Rintanen. A planning algorithm not based on directionnal search. In *Proc. KR-98*, pages 617–624, 1998.

[61] J. Rintanen. Symmetry reduction for SAT representations of transition systems. In *Proc. ICAPS-03*, pages 32–41, 2003.

[62] J. Rintanen. Evaluation strategies for planning as satisfiability. In *Proc. ECAI-04*, pages 682–687, 2004.

[63] J. Rintanen. Asymptotically optimal encodings of conformant planning in QBF. In *Proc. AAAI-07*, 2007.

[64] J. Rintanen, K. Heljanko, and Niemelä. Parallel encodings of classical planning as satisfiability. In *Proc. JELIA-04*, pages 307–319, 2004.

[65] J. Rintanen, K. Heljanko, and Niemelä. Planning as datisfiability : Parallel plans and algorithms for plan search. *Artificial Intelligence*, 170(1213) :1031–1080, 2006.

[66] B. Selman, H. Kautz, and B. Cohen. Noise strategies for improving local search. In *Proc. AAAI-94*, pages 337–343, 1994.

[67] A. van Gelder and F. Okushi. A propositional theorem prover to solve planning and other problems. *AMAI*, 26(1-4) :87–112, 1999.

[68] V. Vidal. *Recherche dans les graphes de planification, satisfiabilité et stratégies de moindre engagement. Les systèmes LCGP et LCDPP*. PhD thesis, IRIT, Université Paul Sabatier, 10 July 2001.

[69] V. Vidal. Le moindre engagement dans une approche de la planification basée sur la procédure de Davis et Putnam. In *Proc. JNPC-02*, pages 239–253, 2002.

[70] V. Vidal. A lookahead strategy for heuristic search planning. In *Proc. ICAPS-04*, pages 150–159, 2004.

[71] V. Vidal and H Geffner. Branching and pruning : An optimal temporal POCL planner based on constraint programming. In *Proc. AAAI-04*, pages 570–577, 2004.

[72] V. Vidal and H. Geffner. Branching and pruning : An optimal temporal POCL planner based on constraint programming. *Artificial Intelligence*, 170(3) :298–335, 2006.

[73] D. Weld. An introduction to least commitment planning. *AI Magazine*, 15(4) :27–61, 1994.

[74] S. Wolfman and D. Weld. The LPSAT engine and its application to resource planning. In *Proc. IJCAI-99*, pages 310–317, 1999.

[75] S. Wolfman and D. Weld. Combining linear programming and satisfiability solving for resource planning. *Knowledge Engineering Review*, 15(1), 2000.

[76] Z. Xing, Y. Chen, and W. Zhang. Maxplan : Optimal planning by decomposed satisfiability and backward reduction. In *Proc. ICAPS-06*, pages 53–56, 2006.

[77] H. L. S. Younes and R. G. Simmons. VHPOP : Versatile heuristic partial order planner. *JAIR*, 20 :405–430, 2003.

[78] H. Zhang. SATO : An efficient propositional prover. In *Proc. CADE-97*, pages 272–275, 1997.

[79] X. Zhao, Y. Chen, and W. Zhang. Optimal planning by maximum satisfiability and accumulative learning. In *Proc. ICAPS-06*, pages 442–447, 2006.