SibylSat : Utiliser SAT comme oracle pour effectuer une recherche gloutonne en planification TOHTN

Gaspard Quenard[1], Damien Pellier[1], Humbert Fioriono[1]

[1] University Grenoble Alpes, LIG

12 mai 2025

Résumé

Cet article présente SibylSat, une nouvelle méthode basée sur SAT conçue pour résoudre efficacement les problèmes HTN Totalement Ordonnés (TOHTN). Contrairement aux planificateurs HTN basés sur SAT qui emploient une stratégie de recherche en largeur d'abord, SibylSat adopte une approche de recherche gloutonne, lui permettant d'identifier les décompositions prometteuses à développer. Le processus de sélection est facilité par une heuristique indépendante du domaine, dérivée de la résolution d'un problème relaxé, qui est également exprimé comme un problème SAT. Nos évaluations expérimentales démontrent que SibylSat surpasse les approches TOHTN basées sur SAT existantes en termes de couverture, de temps d'exécution et de qualité de plan sur la plupart des benchmarks de la Compétition International de Planification (IPC), tout en résolvant un plus grand nombre de problèmes.

Mots-clés

TOHTN, SAT, planification automatique.

1 Introduction

La planification par réseaux de tâches hiérarchiques (HTN) [10] est une technique de planification spécifique qui se concentre sur la décomposition des tâches complexes en sous-tâches plus simples. Contrairement à la planification classique, elle introduit le concept de tâches abstraites (actions de haut niveau qui ne peuvent pas être exécutées directement) et des méthodes de décomposition, lesquelles spécifient un ensemble partiellement ordonné de sous-tâches primitives et de sous-tâches abstraites supplémentaires à exécuter pour réaliser la tâche abstraite. L'objectif d'un planificateur HTN est de transformer une tâche abstraite initiale en un plan, c'est-à-dire une séquence exécutable d'actions. La planification HTN constitue un domaine de recherche majeur et fait désormais partie de la Compétition Internationale de Planification (IPC) [5, 24].

Dans cet article, nous concentrons notre étude sur la planification HTN totalement ordonnée (TOHTN) : une sous-classe particulièrement répandue des problèmes HTN, où les méthodes de décomposition spécifient une liste totalement ordonnée d'actions primitives et de tâches abstraites à exécuter afin de réaliser une tâche abstraite donnée. Les approches dominantes en planification TOHTN consistent soit à utiliser la recherche heuristique, par exemple [13, 14, 15], soit à encoder les problèmes de planification sous forme de problèmes STRIPS afin de tirer parti des améliorations constantes des planificateurs classiques, par exemple [2, 1, 9], ou encore à les encoder en logique propositionnelle, par exemple [6, 23, 4, 21], puis à employer des solveurs hautement optimisés. Cette dernière approche a suscité un regain d'intérêt ces dernières années, en particulier grâce aux planificateurs ayant obtenu d'excellentes performances lors des récentes compétitions internationales de planification. Le travail présenté dans cet article s'inscrit précisément dans cette tendance.

Dans le contexte de la planification TOHTN, la stratégie prédominante parmi les planificateurs SAT actuels consiste à employer une approche de type « recherche en largeur d'abord », limitée par le nombre maximal de décompositions de la tâche abstraite initiale. Concrètement, pour une borne donnée k, l'encodage logique considère uniquement les plans où la tâche initiale peut être décomposée au plus k fois. Si aucune solution n'est trouvée pour cette borne, la valeur de k est augmentée, élargissant ainsi la recherche à des décompositions plus complexes. Ce processus itératif se poursuit jusqu'à la découverte d'une solution satisfaisante ou jusqu'à ce qu'une limite maximale prédéfinie pour k soit atteinte.

Bien que cette approche par recherche en largeur d'abord garantisse la complétude (une solution, si elle existe, sera trouvée par le planificateur), elle s'apparente à une recherche aveugle, potentiellement très inefficace face à des espaces de recherche importants ou à des instances de problèmes complexes. Contrairement à d'autres techniques de planification qui exploitent des informations heuristiques pour orienter la recherche vers des régions prometteuses, les approches SAT actuelles pour la planification TOHTN n'intègrent pas de recherche heuristique.

Dans cet article, nous proposons SibylSat, une nouvelle approche SAT qui met en œuvre une stratégie de recherche gloutonne de type « best-first ». Cette stratégie implique d'étendre sélectivement certaines zones de recherche en fonction d'une heuristique issue de la résolution d'une version relaxée du problème de planification, ce qui permet d'identifier les décompositions prometteuses. Contrairement aux planificateurs TOHTN SAT existants, SibylSat ne développe pas automatiquement toutes les décomposi-

tions en attente lorsqu'aucune solution n'est trouvée. Au contraire, il détermine intelligemment quelles décompositions doivent être approfondies grâce à cette heuristique. Notre approche réduit ainsi considérablement l'espace de recherche et ouvre la voie à l'élaboration de nouvelles heuristiques et techniques pour résoudre les problèmes HTN par encodage SAT. Expérimentalement, nous démontrons que notre stratégie de recherche gloutonne couplée à une fonction heuristique améliore les performances sur la majorité des benchmarks IPC comparativement aux approches SAT TOHTN actuelles

L'article est organisé comme suit : nous présentons d'abord le concept de la planification TOHTN, puis nous décrivons notre planificateur, SibylSat, et enfin, nous comparons SibylSat aux autres planificateurs TOHTN basés sur SAT.

2 Problème de planification HTN

Cet article adopte une description de la planification TOHTN inspirée de la formulation présentée dans [6, 4], qui est une adaptation de la description de la planification HTN proposée dans [12] et adaptée à la planification totalement ordonnée.

2.1 Tâches, actions, méthodes et réseaux de tâches

Le concept central de la planification HTN est la notion de *tâches*. Une *tâche* est caractérisée par un nom et une liste de paramètres. Il existe deux types de tâches : les *tâches primitives* et les *tâches abstraites*. Alors que les tâches primitives affectent directement l'état du monde, les tâches abstraites ne le font pas ; elles doivent être décomposées en tâches primitives via des *méthodes de décomposition* avant de pouvoir être exécutées. Le problème de planification HTN spécifie comment une méthode peut décomposer une tâche abstraite en un *réseau de tâches*, qui est une séquence de tâches (abstraites ou primitives). Un *réseau de tâches primitives* désigne un réseau de tâches ne contenant que des tâches primitives.

Une tâche primitive a est analogue à une action en planification classique et est définie par un triplet $(\mathit{name}(a), \mathit{precond}(a), \mathit{effect}(a))$. Ici, $\mathit{name}(a)$ représente le nom de a, tandis que $\mathit{precond}(a)$ et $\mathit{effect}(a)$ désignent respectivement les ensembles de propositions définissant les préconditions et les effets. Une action a est considérée comme exécutable dans un état s, défini comme un ensemble de propositions décrivant le monde, si et seulement si $\mathit{precond}(a)$ est un sous-ensemble de s. Si O représente l'ensemble des actions et S l'ensemble des états, alors la fonction de transition d'état $\gamma : S \times O \to S$ est définie comme suit : si a est exécutable dans s, alors $\gamma(s, a) = (s \setminus \mathit{effect}^-(a)) \cup \mathit{effect}^+(a))$; sinon, $\gamma(s, a)$ est indéfinie. L'extension de γ aux séquences d'actions, notée $\gamma : S \times O^* \to S$, est définie de manière triviale.

Une méthode m définit comment une tâche abstraite peut être raffinée en un réseau de tâches. Elle est définie par un triplet $(\mathit{name}(m), c, w_m)$, où c est une tâche abstraite et w_m est un réseau de tâches. On appelle c la tâche de m et w_m les sous-tâches de m. Étant donné un réseau de tâches $w = w_1 c w_2$ où c est une tâche abstraite, l'application d'une méthode $m = (\mathit{name}(m), c, w_m)$ à w aboutira au réseau de tâches $w' = w_1 w_m w_2$ (noté $w \to^m w'$). Nous écrivons $w \to^* w'$ s'il existe une séquence de méthodes de décomposition qui transforme w en w'. Par convention, nous définissons $M(c) = \{m = (name(m), c, w_m) \mid m \in M\}$ comme l'ensemble de toutes les méthodes pouvant être appliquées à la tâche abstraite c.

2.2 Problème de planification et solution

Nous définissons un problème de planification TOHTN comme suit :

Definition 1 (Problème de planification TOHTN) *Un problème de planification HTN totalement ordonné P est un tuple $(L, C, O, M, c_I, s_I, g)$ où : L est un ensemble fini de propositions ; C est un ensemble fini de tâches abstraites (ou composées) ; O est un ensemble fini de tâches primitives (ou actions) ; M est un ensemble fini de méthodes de décomposition (ou méthodes) ; $c_I \in C$ est la tâche abstraite initiale ; $s_I \subseteq L$ est l'état initial ; g est l'état objectif (éventuellement vide).*

Résoudre un problème de planification P consiste à trouver un réseau de tâches primitives (ou *plan*) π pouvant être décomposé à partir de la tâche abstraite initiale ($c_I \to^* \pi$) de sorte que π soit exécutable dans l'état initial s_I et atteigne l'objectif g après exécution, c'est-à-dire $g \subseteq \gamma(s_I, \pi)$. Contrairement à la planification classique, une solution à un problème de planification TOHTN n'est pas simplement une séquence de tâches primitives exécutables dans l'état initial et atteignant un état objectif ; elle doit également indiquer les méthodes de décomposition utilisées pour aboutir à cette séquence de tâches primitives. Une telle solution peut être représentée par un arbre appelé *arbre de décomposition* [12], qui montre l'ensemble des étapes ayant conduit au raffinement d'une tâche abstraite en un réseau de tâches.

Definition 2 (Arbre de décomposition) *Un arbre de décomposition (DT) pour un réseau de tâches w dans un problème $P = (L, C, O, M, c_I, s_I, g)$ est un arbre $T = (N, E)$ avec :*

- *N - Un ensemble de nœuds étiquetés par une tâche primitive, une tâche abstraite ou une méthode.*
- *$E : N \to N^*$ - La fonction d'arêtes qui associe à chaque nœud une liste ordonnée d'enfants $\langle e_1, e_2, ..., e_k \rangle$.*
- *Pour tout nœud interne n étiqueté par une tâche abstraite c, $|E(n)| = 1$ et $E(n) = \langle n' \rangle$ où n' est étiqueté par une méthode $m \in M(c)$.*
- *Pour tout nœud interne n étiqueté par une méthode m, considérons $\langle t_1, t_2, \ldots, t_k \rangle$ les sous-tâches de m. Alors $|E(n)| = k$ et chaque enfant e_i de $E(n)$ est étiqueté par la tâche t_i pour tout $e_i \in E(n) = \langle e_1, e_2, \ldots, e_k \rangle$*
- *Pour la séquence de feuilles $L = \langle n^l_1, n^l_2, \ldots, n^l_k \rangle$, chaque feuille est étiquetée par une tâche primitive*

ou abstraite et, si l'on considère la séquence correspondante de tâches $\langle t_1, t_2, \ldots, t_k \rangle$, alors on a $w = \langle t_1, t_2, \ldots, t_k \rangle$.

Nous pouvons maintenant définir formellement une solution sous forme d'arbre de décomposition :

Definition 3 (Solution par arbre de décomposition) *Soit $P = (L, C, O, M, c_I, s_I, g)$ un problème de planification. Considérons T_{sol} l'arbre de décomposition pour le réseau de tâches π du problème P. L'arbre de décomposition T_{sol} est une solution pour P si et seulement si il satisfait les conditions suivantes :*

1. *La racine de T_{sol} est la tâche abstraite initiale c_I.*
2. *π ne contient que des tâches primitives.*
3. *π est exécutable dans l'état initial s_I.*
4. *π atteint l'objectif g après exécution.*

3 Arbre de décomposition de chemin et planificateurs SAT

Puisque résoudre un problème de planification HTN équivaut à trouver un arbre de décomposition (DT) qui satisfait certaines caractéristiques (cf. Definition 3), une approche consiste à explorer tous les DT possibles enracinés par la tâche abstraite initiale c_I pour un problème donné P. Cependant, l'espace de recherche représentant tous les DT possibles est très vaste pour la plupart des domaines et infini pour les domaines récursifs (domaines où une tâche abstraite peut être obtenue par des décompositions issues de la même tâche abstraite). Pour résoudre ce problème, [6, 23] ont proposé de créer des structures représentant un sous-ensemble de tous les DT possibles afin de vérifier si elles contiennent des solutions valides. Si aucune solution n'est trouvée, ces structures peuvent être étendues pour inclure d'autres DT. Ces structures sont conçues de manière à garantir qu'elles puissent être développées pour inclure tout DT possible enracinés en par la tâche abstraite initiale pour un problème donné. Ici, nous introduisons un équivalent isomorphe de leurs structures :

Definition 4 (Arbre de Décomposition de Chemin) *Un arbre de décomposition de chemin (PDT) pour un problème $P = (L, C, O, M, c_I, s_I, g)$ est un arbre $\Gamma = (N, E)$ avec :*

- *N - Un ensemble de nœuds étiquetés par une tâche primitive, une tâche abstraite ou une méthode.*
- *$E : N \to N^*$ - La fonction d'arêtes qui associe à chaque nœud une liste ordonnée d'enfants $\langle e_1, e_2, ..., e_k \rangle$.*

On note le nœud racine du PDT par r_Γ. On dit qu'un PDT est bien formé si et seulement si :

- *Le nœud racine du PDT r_Γ est étiqueté par la tâche abstraite initiale du problème c_I.*
- *Pour tout nœud interne n étiqueté par une tâche abstraite c, $|E(n)| = |M(c)|$ et $\forall m_i \in M(c), \exists e_i \in E(n)$ tel que e_i est étiqueté par la méthode m_i.*
- *Pour tout nœud interne n étiqueté par une méthode m, considérons $\langle t_1, t_2, \ldots, t_k \rangle$ les sous-tâches de m, alors $|E(n)| = k$ et e_i est étiqueté par la tâche t_i pour tout $e_i \in E(n) = \langle e_1, e_2, \ldots, e_k \rangle$.*
- *Pour toute feuille n^l, n^l est étiquetée par une tâche primitive ou une tâche abstraite.*

La figure 1 illustre un PDT pour un problème $P = (L, C, O, M, c_I, s_I, g)$ à un certain niveau de décomposition, et met en évidence un DT pour le réseau de tâches $\langle A_2, A_1, A_3, A_4 \rangle$ comme un sous-arbre de ce PDT. Ce DT est une solution pour P si et seulement si $\pi = \langle A_2, A_1, A_3, A_4 \rangle$ est exécutable dans s_I et $g \subseteq \gamma(s_I, \pi)$.

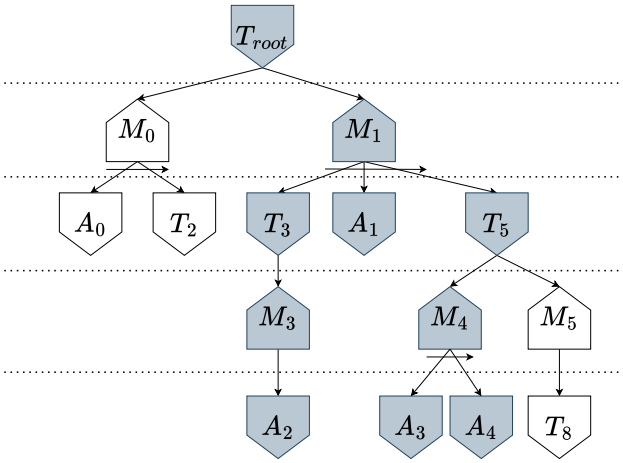

FIGURE 1 – Exemple illustrant un PDT à un certain niveau de décomposition pour un problème $P = (L, C, O, M, c_I, s_I, g)$, où $T_i \in C$, $M_i \in M$ et $A_i \in O$. On remarque que ce PDT ne contient pas tous les DT possibles, car les tâches abstraites T_2 et T_8 ne sont pas développées. Un DT solution potentiel est mis en évidence en gris.

La recherche d'une solution au sein d'un PDT donné est l'approche employée par tous les planificateurs TOHTN basés sur SAT actuels [23, 21, 6, 4]. L'idée clé consiste à encoder un PDT (ou une structure équivalente isomorphe) dans une formule qui est satisfaisable si et seulement s'il existe une solution dans le PDT. Si la formule est satisfaisable, la solution peut être extraite de l'assignation satisfaisante. Si aucune solution n'est trouvée dans le PDT, celui-ci peut être étendu en développant un nombre quelconque de ses *nœuds non développés* (feuilles étiquetées par des tâches abstraites). Pour chaque nœud sélectionné étiqueté par une tâche abstraite c, son processus d'expansion suit deux étapes : d'abord, on ajoute toutes les méthodes applicables $M(c)$ comme enfants ; ensuite, pour chaque méthode, on ajoute sa séquence ordonnée de sous-tâches en tant qu'enfants. Les encodages existants étendent le PDT en développant tous ses nœuds non développés. Si le PDT initial est initialisé avec un seul nœud représentant la tâche abstraite initiale, [6] ont montré que, en suivant cette ap-

proche d'expansion, le $j^{\text{ème}}$ PDT capturera tous les DT possibles avec une profondeur de décomposition maximale de j. Ainsi, cette approche correspond à une recherche en "largeur d'abord" selon la profondeur de décomposition et est, par conséquent, complète.

4 SibylSat planner

4.1 Approche de planification

Comme les autres planificateurs TOHTN basés sur SAT, SibylSat utilise un PDT comme espace de recherche et suit une procédure standard alternant entre l'expansion de l'espace de recherche, l'encodage et l'invocation d'un solveur SAT pour trouver un plan solution. Cependant, une différence majeure réside dans la stratégie d'expansion, qui ne suit pas une approche en largeur d'abord. Au lieu de cela, SibylSat sélectionne des tâches abstraites prometteuses à développer en priorité.

La procédure est illustrée sous forme de pseudo-code dans l'algorithme 1. L'algorithme commence par initialiser le PDT avec un seul nœud représentant la tâche abstraite initiale du problème. Ensuite, le processus alterne entre deux phases jusqu'à ce qu'une solution soit trouvée : l'une où SibylSat recherche un DT solution dans le PDT (cette étape servant de condition d'arrêt), et l'autre où il étend le PDT en explorant en priorité les tâches abstraites les plus prometteuses. Pour ce faire, il cherche à identifier et à développer un *DT prometteur* dans le PDT, où "prometteur" désigne un DT dont l'expansion (c'est-à-dire le développement de tous ses nœuds en attente) pourrait conduire à un DT solution. Le choix du DT prometteur à développer est effectué en recherchant un DT solution dans une relaxation du PDT où les feuilles représentant des tâches abstraites sont considérées comme des actions. Le DT solution trouvé dans ce PDT relaxé sert alors d'heuristique pour identifier les nœuds en attente à développer dans le PDT afin d'atteindre une solution. Plus précisément, toute tâche abstraite faisant partie du plan du DT solution relaxé verra sa feuille correspondante dans le PDT développée. La recherche d'un DT solution dans le PDT, ainsi que celle d'un DT prometteur dans sa version relaxée, est effectuée à l'aide d'un solveur SAT.

Une condition nécessaire (mais non suffisante) pour garantir que notre planificateur soit complet est que tout sous-arbre d'un DT solution doit également être reconnu comme un DT prometteur dans un PDT relaxé. Ne pas respecter cette condition pourrait conduire à ignorer des solutions valides. Par conséquent, lors de la relaxation du PDT, les préconditions et effets attribués à une tâche abstraite doivent être soigneusement inférés pour satisfaire cette exigence. L'approche utilisée pour les déduire sera détaillée dans une section ultérieure. Il est important de noter que cette implication n'est pas bidirectionnelle : un DT prometteur ne conduit pas nécessairement à un DT solution lorsqu'il est davantage développé. Plus nous sommes capables de représenter fidèlement les tâches abstraites sous forme d'actions (c'est-à-dire d'inférer correctement leurs préconditions et effets), plus nous nous rapprochons de cette équivalence.

L'algorithme 1 est qualifié de "glouton" car il étend le PDT en utilisant le premier DT solution relaxé trouvé lors de la phase d'expansion, même si plusieurs DT solutions relaxés existent dans le PDT. Cette méthode est complète pour les domaines de planification non récursifs car le nombre de DT est fini, et notre algorithme finit par tous les explorer à mesure qu'il étend le PDT. Cependant, cette approche est généralement non terminante pour les domaines récursifs, car l'algorithme peut continuer à développer des DT dans le PDT sans jamais aboutir à un DT solution. Nous expliquerons dans une section ultérieure comment modifier l'algorithme 1 pour garantir sa terminaison.

Algorithm 1 SibylSat Planner

```
1:  procedure SibylSat(P = (L, C, O, M, c_I, s_I, g))
2:      PDT ← InitializePDT(P)
3:      return Greedy(PDT)
4:  end procedure
5:  procedure Greedy(PDT)
6:      DT_sol ← FindSolution(PDT)
7:      if DT_sol ≠ ∅ then
8:          return DT_sol
9:      end if
10:     DT_relaxed ← FindPromisingDT(PDT)
11:     PDT ← ExpandPDT(PDT, DT_relaxed)
12:     return Greedy(PDT)
13: end procedure
```

4.2 Exemple

Pour illustrer notre algorithme, nous détaillons comment le PDT d'un problème P est développé par notre planificateur dans la Figure 1. SibylSat est initialisé en créant un PDT avec un seul nœud racine, T_{root}, représentant la tâche abstraite initiale du problème. Il alterne ensuite entre une phase de recherche et une phase d'expansion jusqu'à ce qu'un DT solution soit trouvé.

Dans la phase de recherche initiale, comme T_{root} est une tâche abstraite, le PDT ne contient pas encore de DT solution, ce qui déclenche la première phase d'expansion. Lors de cette phase, SibylSat applique la technique de relaxation pour transformer toutes les feuilles correspondant à des tâches abstraites, dans ce cas T_{root}, en actions. Une recherche est ensuite effectuée dans le PDT relaxé, identifiant le DT relaxé correspondant au plan $\langle T_{root} \rangle$ comme solution. Ce DT est alors développé dans le PDT, aboutissant à un PDT étendu encapsulant les trois premières couches de la Figure 1, avant de revenir à la phase de recherche.

La deuxième phase de recherche échoue de nouveau à trouver un DT solution dans le PDT, ce qui entraîne une nouvelle phase d'expansion. Au cours de cette phase, le PDT est à nouveau relaxé en considérant les tâches abstraites feuilles T_2, T_3 et T_5 comme des actions et en cherchant un DT solution dans ce PDT relaxé. Il est intéressant de noter que, dans ce PDT relaxé, deux DT solutions potentiels émergent : l'un impliquant le réseau de tâches $\langle A_0, T_2 \rangle$ et un autre comprenant $\langle T_3, A_1, T_5 \rangle$. En supposant que ces réseaux de tâches soient tous deux exécutables dans l'état ini-

tial du problème et permettent d'atteindre l'objectif après exécution, le solveur SAT sélectionne l'un d'eux pour un développement ultérieur. Dans cet exemple, le solveur SAT choisit le DT du réseau de tâches $\langle T_3, A_1, T_5 \rangle$, incitant SibylSat à développer les nœuds correspondants T_3 et T_5, ce qui conduit à l'ensemble du PDT présenté dans la Figure 1, avant de revenir à la phase de recherche.

Lors de cette phase de recherche dans le PDT, SibylSat identifie le DT solution associé au plan $\langle A_2, A_1, A_3, A_4 \rangle$, conduisant à la terminaison réussie du planificateur. Cependant, si cette solution s'avérait non exécutable dans l'état initial, l'absence d'autres DT solutions possibles aurait déclenché une nouvelle phase d'expansion du PDT. Dans ce cas, si les DT des réseaux de tâches $\langle A_0, T_2 \rangle$ et $\langle A_2, A_1, T_8 \rangle$ étaient des DT solutions relaxés, alors le solveur SAT aurait retourné l'un des deux, ce qui aurait conduit au développement soit du nœud T_2, soit du nœud T_8.

5 Recherche d'un DT solution dans un PDT

Lors de la phase de recherche, notre planificateur doit déterminer si un PDT contient un DT solution en tant que sous-arbre. Pour ce faire, l'approche des planificateurs TOHTN basés sur SAT consiste à créer une formule SAT qui est satisfiable si et seulement si un DT solution existe dans le PDT. Étant donné que notre espace de recherche est structurellement congruent avec ceux proposés par [21, 23, 6, 4], nous pouvons utiliser n'importe lequel des encodages introduits dans ces travaux pour rechercher un DT solution. Nous avons choisi d'utiliser l'encodage du planificateur Lilotane, en particulier parce qu'il est compatible avec un solveur SAT incrémental. Cela permet à notre solveur de conserver les connaissances acquises lors des itérations précédentes lorsqu'il recherche des DT solutions et des DT solutions relaxés. Comme l'encodage utilisé pour rechercher un DT solution dans un PDT est identique à celui proposé par Lilotane, nous ne détaillons pas dans cet article les règles SAT spécifiques employées pour rechercher un DT solution.

6 Expansion du PDT

Dans cette section, nous présentons la phase d'expansion du PDT. Cette phase intervient lorsqu'aucun DT solution n'existe dans le PDT actuel (c'est-à-dire lorsque la procédure de recherche a échoué à en trouver un). Dans ce cas, certaines feuilles en attente doivent être développées pour permettre au PDT de capturer un ensemble plus large de DT. L'idée principale de notre phase d'expansion est de trouver un DT qui soit un sous-arbre d'un DT solution. Si un tel DT peut être identifié, son développement permettra d'étendre le PDT jusqu'à inclure un DT solution en tant que sous-arbre. Cependant, identifier un DT menant à un plan primitif exécutable n'est pas faisable, car cela nécessiterait de connaître à l'avance un DT solution pour le problème. Ainsi, nous nous intéressons à une approximation. Cette approximation consiste à identifier un DT prometteur : un DT qui pourrait conduire à un DT solution une fois développé. Nous définissons un DT comme "prometteur" si son réseau de tâches relaxé est exécutable dans l'état initial et atteint l'objectif après exécution. Pour relaxer un réseau de tâches, nous convertissons toutes ses tâches abstraites en actions avec des préconditions et des effets basés sur les décompositions possibles de la tâche abstraite correspondante. Trouver un DT prometteur revient ainsi à rechercher un DT solution dans un PDT relaxé où toutes les feuilles représentant des tâches abstraites sont converties en actions.

Nous nous intéressons maintenant au processus d'inférence des préconditions et des effets d'une tâche abstraite. Comme détaillé dans l'approche de planification, une condition nécessaire pour que notre planificateur soit complet est que tous les sous-arbres d'un DT solution soient reconnus comme des DT prometteurs. Par conséquent, les préconditions et effets inférés pour une tâche abstraite doivent être valides pour toutes les raffinements possibles de cette tâche en réseaux de tâches primitifs.

Nous commençons par expliquer l'inférence des préconditions d'une tâche abstraite. Ces préconditions doivent englober les faits nécessaires à tout raffinement possible de la tâche abstraite. [20] ont qualifié ces préconditions de "préconditions obligatoires", un concept utilisé dans plusieurs planificateurs [21, 16] pour éliminer de l'espace de recherche les DT qui ne mènent pas à une séquence de tâches primitives exécutable. Dans notre planificateur, nous employons l'algorithme d'inférence des préconditions obligatoires proposé par [21].

Considérons maintenant les effets d'une tâche abstraite. Contrairement aux actions, dont les effets sont explicitement définis, les tâches abstraites introduisent une incertitude quant à l'état résultant. En effet, pour une action a, la transition $\gamma(s, a) = (s \setminus \mathit{effect}^-(a)) \cup \mathit{effect}^+(a)$ définit précisément l'état après exécution. En revanche, il est difficile de définir l'état post-exécution d'une tâche abstraite, car elle peut être décomposée en plusieurs réseaux de tâches primitives ayant des effets post-exécution différents. Notre solution consiste à calculer l'ensemble $\mathit{poss\text{-}effect}^+(t)$ (respectivement $\mathit{poss\text{-}effect}^-(t)$) d'une tâche abstraite, correspondant à tous les faits positifs (respectivement négatifs) pouvant être causés par un raffinement de cette tâche abstraite.

Si nous parvenons à calculer ces deux ensembles, nous pouvons alors définir une surestimation de l'état post-exécution d'une tâche abstraite t. Cela est réalisé en considérant l'exécution de t comme menant à un état non déterministe, défini par l'ajout d'un sous-ensemble quelconque de $\mathit{poss\text{-}effect}^+(t)$ et la suppression d'un sous-ensemble quelconque de $\mathit{poss\text{-}effect}^-(t)$ à l'état pré-exécution.

[4] ont montré que le calcul exact des effets possibles d'une tâche abstraite est un problème PSPACE- ou EXPTIME-complet, selon la structure hiérarchique de décomposition de la tâche concernée. Toutefois, il est possible d'obtenir une surestimation en temps polynomial. Pour notre algorithme, nous calculons une surestimation des effets possibles d'une tâche abstraite (notée $\mathit{poss\text{-}effect}_*$) avec la formule suivante :

$$\textit{poss-effect}_*^+(t) = \bigcup_{t' \in subtasks(m), m \in M(t)} \textit{poss-effect}_*^+(t')$$

$$\textit{poss-effect}_*^-(t) = \bigcup_{t' \in subtasks(m), m \in M(t)} \textit{poss-effect}_*^-(t')$$

Si t est une action, alors :

$$\textit{poss-effect}_*^+(t) = \textit{effect}^+(t)$$
$$\textit{poss-effect}_*^-(t) = \textit{effect}^-(t)$$

Si t est récursive, nous évitons une récursion infinie en supprimant t de l'ensemble des sous-tâches possibles de toutes les méthodes.

Il convient de noter que l'état post-exécution non déterministe d'une tâche abstraite, tel que défini ci-dessus, constitue souvent une surestimation de l'état post-exécution qu'un raffinement concret spécifique de la tâche abstraite pourrait réellement produire, car il englobe les effets de divers raffinements. Bien que [19] aient proposé une méthode permettant de généraliser les effets possibles d'une tâche abstraite en les différenciant selon les résultats de chaque raffinement, cette méthode est trop coûteuse en termes de calcul. Néanmoins, il reste possible d'affiner l'état post-exécution non déterministe d'une tâche abstraite en éliminant les incohérences résultant de la combinaison des effets de différents raffinements.

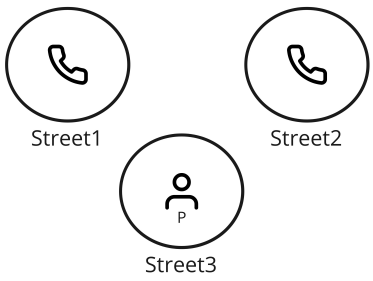

FIGURE 2 – État du monde simplifié.

Considérons l'exemple illustré dans la Figure 2. Cette figure représente un état du monde avec trois emplacements : Street1, Street2 et Street3. Une personne, notée 'P', est située à Street3 et doit appeler un taxi. Considérons une tâche abstraite $call\_taxi()$, dont l'objectif est d'appeler un taxi après s'être rendu à un emplacement où se trouve une cabine téléphonique. Dans cet exemple, il existe une cabine téléphonique à la fois à Street1 et Street2, de sorte que les effets possibles de la tâche abstraite $call\_taxi()$ incluront les prédicats $at(P, Street1)$ et $at(P, Street2)$. Si cette tâche est traitée comme une action directe, l'application de notre formule pour calculer son état post-exécution pourrait aboutir à une incohérence où la personne se retrouve simultanément à Street1 et Street2. Un tel état incohérent pourrait conduire à la découverte de DT solutions relaxés qui sont impraticables, c'est-à-dire qui ne peuvent aboutir à un DT solution valide une fois développés. Cela entraîne souvent le développement de nœuds inutiles à la recherche d'un DT solution valide.

Notre approche pour réduire ces incohérences repose sur l'utilisation de mutex. Les mutex définissent les ensembles de faits qui ne peuvent pas coexister dans le même état atteignable. Dans notre exemple, les prédicats $at(P, Street1)$ et $at(P, Street2)$ sont en mutex, car une personne ne peut pas être à deux endroits en même temps. Pour inférer ces mutex, nous utilisons l'algorithme proposé par [11], qui détecte des groupes de mutex, c'est-à-dire des ensembles de prédicats où chaque prédicat est en mutex avec tous les autres du même groupe. En utilisant ces mutex, nous améliorons la qualité de l'état post-exécution des tâches abstraites en empêchant la cooccurrence d'effets incompatibles.

En pratique, nous appliquons une contrainte "au plus un" dans notre encodage SAT pour empêcher les effets mutuellement exclusifs d'une tâche abstraite de se produire simultanément. Il existe plusieurs méthodes pour encoder ces contraintes dans SAT, comme discuté par [18]. Nos expériences montrent que, avec le solveur Glucose [3], l'encodage par paires classiques semble légèrement plus performant que les autres encodages testés, bien qu'il consomme plus de mémoire.

Nos résultats expérimentaux indiquent que, dans de nombreux domaines, l'utilisation des mutex permet de trouver des DT solutions abstraits de meilleure qualité et, par conséquent, de développer moins de nœuds dans le PDT avant d'aboutir à un DT solution.

6.1 Recherche d'un DT solution dans un PDT relaxé

Une fois que le PDT a été relaxé afin de remplacer toutes les tâches abstraites en feuilles par leurs actions correspondantes, le processus de recherche devient similaire à celui du PDT non relaxé, à une différence majeure près : les effets des tâches abstraites sont non déterministes. Cette incertitude signifie qu'un effet listé dans $\textit{poss-effect}_*(c)$ peut ne pas nécessairement se produire, même si c est exécutée. Par conséquent, lorsqu'on utilise un encodage SAT pour rechercher un DT solution dans un PDT relaxé, les effets d'une tâche abstraite en tant que feuille ne peuvent pas être encodés de la même manière que les effets d'une action, qui sont représentés par $action \implies \textit{eff}$. Au lieu de cela, pour chaque effet possible d'une tâche abstraite en tant que feuille, nous devons encoder le fait que si cette tâche est incluse dans un DT solution, alors chacun de ses effets *peut* se produire dans son état post-exécution.

Lilotane intègre déjà des règles permettant d'encoder cette incertitude dans l'état post-exécution d'une tâche abstraite, appelées *frame axioms*. Ces axiomes sont des règles qui garantissent que si la valeur d'un prédicat change, alors une tâche abstraite ou une action spécifique ayant cet effet a nécessairement été exécutée. En utilisant ces frame axioms pour encoder les effets possibles d'une tâche abstraite, nous prenons correctement en compte leurs effets non déterministes.

Par exemple, supposons qu'une tâche t puisse potentiellement modifier un prédicat p, c'est-à-dire que p est un effet possible de t. Si le prédicat p change de valeur entre l'état avant et l'état après l'exécution de t, alors la tâche t doit nécessairement être incluse dans le DT solution.
Cela peut être formalisé par le frame axiom suivant :

$$\neg p\_avant\_t \wedge p\_apres\_t \implies t$$

Cependant, l'implication n'est pas réciproque. Si la tâche t est incluse dans le DT solution, cela ne garantit pas que le prédicat p devienne vrai. Ainsi, le non-déterminisme des effets des tâches abstraites est préservé.

6.2 Développement des feuilles du PDT

Enfin, nous expliquons comment nous développons les nœuds en attente d'un PDT en fonction d'un DT solution trouvé dans un PDT relaxé. Dans le PDT relaxé, toutes les feuilles représentant des tâches abstraites sont considérées comme des actions. Par conséquent, le plan π dans le DT solution peut inclure des tâches abstraites. Pour chacune de ces tâches abstraites, nous développons le nœud correspondant dans le PDT ainsi que tous ses nœuds enfants immédiats correspondant aux méthodes applicables. Cette approche garantit que toutes les feuilles du PDT contiennent soit des tâches primitives, soit des tâches abstraites. Par exemple, comme illustré dans la Figure 1, si le plan du DT solution pour le PDT relaxé est $\langle A_0, T_2 \rangle$, alors le nœud contenant T_2 ainsi que ses méthodes enfants immédiates seront développés.
Il est à noter que, dans notre implémentation, nous utilisons l'espace de recherche proposé par le planificateur Lilotane, où chaque nœud contient toutes les tâches abstraites pouvant apparaître simultanément à un certain niveau de raffinement. Afin d'adhérer à l'encodage incrémental fourni par Lilotane, où certaines règles dépendent de l'ensemble complet des opérations présentes dans un nœud, nous développons toutes les tâches abstraites du même nœud lorsqu'un tel nœud est étendu. Une future amélioration possible pourrait consister à modifier l'encodage afin de permettre le développement sélectif des seules tâches abstraites identifiées dans le DT solution du PDT relaxé.

7 Assurer la complétude pour les domaines récursifs

Comme la description actuelle de notre planificateur suit une approche gloutonne en "best-first" plutôt qu'une recherche en largeur d'abord, utilisée par les autres planificateurs TOHTN basés sur SAT, cette approche peut souffrir d'une non-termination dans les domaines récursifs.
Une solution possible pour atténuer ce problème consisterait à limiter la profondeur maximale de décomposition du PDT en utilisant une approche d'approfondissement itératif, garantissant que tous les DT avec une profondeur de décomposition allant jusqu'à k soient explorés avant d'augmenter progressivement cette limite. En pratique, deviner la profondeur initiale de décomposition et la manière de l'incrémenter peut être complexe, car cela dépend de la structure hiérarchique du domaine ainsi que des caractéristiques du problème (objets, tâche abstraite initiale et état initial).
Notre approche adopte une technique plus ciblée qui s'appuie directement sur la structure du domaine. Nous observons que, pour chaque tâche abstraite non récursive, le processus de décomposition répétée est fini. Par conséquent, il suffit uniquement de limiter la décomposition des tâches abstraites récursives pour assurer la complétude. Nous appliquons l'approche suivante : pour chaque tâche abstraite récursive t dans le PDT, nous empêchons l'un de ses enfants ou enfants transitifs d'être identique à t. Plus précisément, si un nœud étiqueté par une méthode m est un enfant ou un descendant d'un nœud étiqueté par une tâche abstraite t, et que l'une des sous-tâches de la méthode m est t, alors nous excluons m du PDT. Cette technique garantit que l'expansion exhaustive de notre PDT reste finie. Si aucun DT solution n'est trouvé lorsque le PDT est complètement développé avec ces restrictions, nous intégrons alors toutes les méthodes m exclues ainsi que leurs sous-tâches dans le PDT et nous répétons le processus. Cette approche permet à notre algorithme d'assurer la complétude. [16] ont proposé une technique très similaire pour éviter les boucles infinies dans leur planificateur TOHTN HyperTensionN.
Empiriquement, cette méthodologie s'est avérée efficace. Notre planificateur a été capable de résoudre des problèmes issus des benchmarks de la Compétition Internationale de Planification (IPC) de 2020 et 2023 sans jamais avoir besoin d'intégrer une seule fois les méthodes exclues dans le PDT.

8 Evaluation

8.1 Planificateurs

Nous avons comparé SibylSat à d'autres planificateurs TOHTN basés sur SAT à l'état de l'art afin d'évaluer ses performances. L'évaluation a porté sur trois planificateurs : SibylSat [1], notre planificateur utilisant l'approche garantissant la complétude avec le solveur Glucose [3] ; Lilotane [21], un planificateur TOHTN SAT Lifted, arrivé deuxième lors de la Compétition Internationale de Planification (IPC) 2020 ; et pandaPIsatt-1iB [4], une version améliorée du planificateur TOHTN SAT original totSAT [6]. Les expériences ont été réalisées sur un système équipé d'un processeur Intel Core i7-12700H et de 32 Go de RAM. Chaque instance de problème disposait d'un temps d'exécution maximal de 10 minutes. L'évaluation a été effectuée sur tous les benchmarks TOHTN proposés lors des IPC 2020 et IPC 2023.

8.2 Métriques d'évaluation

Les performances des planificateurs sont évaluées à l'aide de deux métriques différentes :

8.2.1 Score IPC

Le score IPC mesure la rapidité avec laquelle un planificateur résout un problème. Ce score varie de 0 à 1 pour chaque problème, avec 1 indiquant que le planificateur a trouvé une

1. Le code source est disponible à l'adresse : https ://github.com/gaspard-quenard/sibylsat

solution en moins d'une seconde et 0 correspondant à un dépassement du temps limite. Il est calculé comme suit :

$$\text{Score IPC} = \begin{cases} 0 & \text{Si pas de plan trouvé} \\ \min\left(1, 1 - \frac{\log(t)}{\log(T)}\right) & \text{Sinon} \end{cases}$$

où T est le temps d'exécution maximal autorisé pour trouver un plan et t est le temps (en secondes) mis par le planificateur pour trouver un plan. L'utilisation des logarithmes dans le calcul permet de mieux discriminer les performances lorsque les temps de résolution sont faibles ; ainsi, un planificateur trouvant une solution significativement plus vite qu'un autre recevra un score proportionnellement bien meilleur, encourageant la recherche de solutions rapides.

8.2.2 Score de qualité

Le score de qualité mesure le makespan (c'est-à-dire le nombre total d'actions dans un plan). Il varie de 0 à 1, avec 1 correspondant au plan le plus court et 0 signifiant qu'aucune solution n'a été trouvée. Il est défini comme suit :

$$\text{Quality Score} = \begin{cases} 0 & \text{Si pas de plan trouvé} \\ \frac{C^{ref}}{C} & \text{Sinon} \end{cases}$$

où C est le makespan du plan trouvé par le planificateur et C^{ref} est le meilleur makespan parmi tous les planificateurs évalués.

8.3 Résultats

Un aperçu des résultats pour la couverture, le score IPC et le score de qualité est présenté dans le tableau 1. Nous observons que, parmi les 26 domaines des benchmarks, notre planificateur obtient un meilleur score IPC dans 19 d'entre eux et dépasse également les autres planificateurs en termes de couverture et de score de qualité. Notons que le score de qualité est calculé sur la base du premier DT solution trouvé et n'inclut donc pas les procédures d'amélioration de plan, comme celle proposée par Lilotane, qui permet de trouver le DT solution avec le plan le plus court dans un PDT au prix d'un temps d'exécution plus élevé.

Pour vérifier si la stratégie d'expansion proposée par SibylSat permet effectivement d'explorer une plus petite partie de l'espace de recherche avant de trouver un DT solution, nous comparons dans la Figure 3 le nombre de méthodes développées avant de trouver un DT solution par Lilotane et SibylSat, étant donné que les deux planificateurs utilisent la même structure pour représenter l'espace de recherche. La figure montre que notre approche réduit généralement le nombre de méthodes développées avant de trouver un DT solution dans la plupart des domaines. Dans certains domaines, tels que Towers et Childsnack, le nombre de méthodes développées par notre planificateur est similaire à celui de Lilotane, car Lilotane développe déjà le nombre minimal de nœuds dans sa structure pour trouver un DT solution. Le seul benchmark où notre planificateur développe plus de méthodes que Lilotane est le domaine AssemblyHierarchical (indiqué par les marqueurs en étoile violette dans la Figure 3). Cela se reflète dans les résultats, car il s'agit du seul benchmark où le score IPC de SibylSat est inférieur à celui de Lilotane et de PandaPIsatt-1iB. Une analyse du domaine AssemblyHierarchical révèle la cause sous-jacente de ce résultat : la plupart des tâches abstraites de ce domaine sont récursives et fortement interconnectées, ce qui entraîne un ensemble d'effets possibles plus vaste pour chaque tâche. En raison de cette complexité, les effets possibles de chaque tâche abstraite sont étendus et conduisent souvent notre planificateur à identifier des DT solutions relaxés peu pertinents. Une amélioration de notre algorithme d'inférence des effets possibles des tâches abstraites pourrait probablement aider à résoudre ce type de problème.

Domain	# instances	SibylSat	Lilotane	PandaPIsatt-1iB
AssemblyHierarchical	30	2.61	3.89	**4.05**
Barman-BDI	20	**16.42**	15.40	15.05
Blocksworld-GTOHP	30	**25.66**	22.56	21.11
Blocksworld-HPDDL	30	**6.74**	0.80	2.91
Childsnack	30	**27.09**	26.78	21.56
Depots	30	**25.49**	22.05	24.80
Elevator-Learned	147	**146.91**	114.23	137.45
Entertainment	12	8.49	2.46	**11.53**
Factories-simple	20	**6.12**	3.77	5.99
Freecell-Learned	60	**7.31**	5.26	6.25
Hiking	30	**24.87**	22.07	18.89
Lamps	30	15.40	0	**17.31**
Logistics-Learned	80	**60.94**	28.48	49.00
Minecraft-Player	20	**2.66**	2.49	1.43
Minecraft-Regular	59	**32.78**	29.55	29.82
Monroe-FO	20	18.38	**19.09**	11.56
Monroe-PO	20	17.07	**18.21**	9.47
Multiarm-Blocksworld	74	**11.66**	1.87	9.06
Robot	20	**10.92**	10.55	10.75
Rover-GTOHP	30	**21.74**	18.05	18.54
Satellite-GTOHP	20	**14.70**	12.39	14.64
SharpSAT	21	**10.21**	8.35	8.61
Snake	20	**19.79**	19.41	16.57
Towers	20	**9.51**	8.11	5.98
Transport	40	35.05	31.18	**35.49**
Woodworking	30	26.41	**30**	22.01
Coverage	948	**706**	583	664
Normalized coverage	26	**18.97**	15.45	17.61
IPC score		**604.93**	477.0	529.83
Normalized IPC score		**16.06**	13.22	13.93
Quality score		**689.53**	520.83	606.49
Normalized quality score		**18.37**	14.17	15.91

TABLE 1 – Performance de chaque planificateur sur les benchmarks IPC 2020 et IPC 2023. Chaque cellule contient le score IPC pour un domaine spécifique. La couverture totale, le score de qualité total et le score IPC total, ainsi que leurs valeurs normalisées par domaine, sont affichés dans les six dernières lignes.

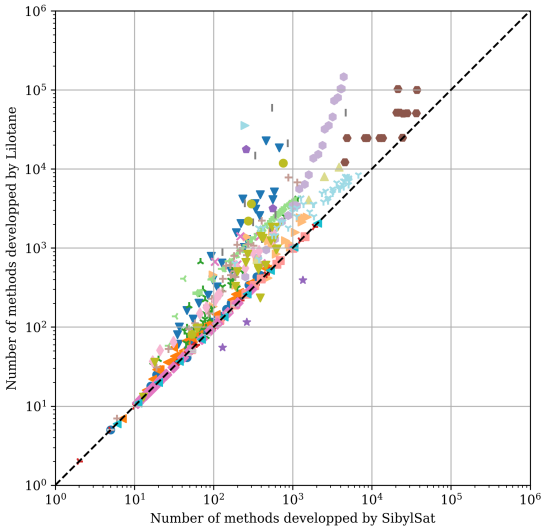

FIGURE 3 – Comparaison du nombre de méthodes développées par SibylSat et Lilotane. Un marqueur différent est utilisé pour chaque benchmark.

9 Travaux connexes

L'utilisation de planificateurs HTN basés sur SAT a été introduite pour la première fois par Mali et Kambhampati en 1998 [17], bien que leur représentation des problèmes HTN différait considérablement des formalismes actuels et que leur encodage ne permettait pas de gérer les domaines récursifs. Malgré cette première exploration, il s'est écoulé près de deux décennies sans recherches spécifiques sur la traduction des problèmes de planification HTN en logique propositionnelle. Ce n'est qu'en 2018 que totSAT [6] a été introduit, proposant une traduction SAT pour la planification TOHTN qui surpassait les autres planificateurs HTN à l'état de l'art, relançant ainsi l'intérêt pour cette approche. Contrairement à la planification classique, où les encodages sont étendus itérativement en fonction de la longueur du plan final, totSAT utilise une recherche en largeur d'abord pour étendre les encodages en fonction de la profondeur de la hiérarchie. À la suite de ce travail, Behnke et al. ont affiné leur approche pour gérer la planification HTN partiellement ordonnée [7], la recherche de plans optimaux [8], et ont proposé des techniques pour réduire l'espace de recherche et améliorer leur encodage [4].

Parallèlement à totSAT, Schreiber et al. ont proposé un nouvel encodage qui fut le premier à exploiter la résolution SAT incrémentale pour les problèmes HTN [22]. Cette approche a ensuite été améliorée avec le développement du planificateur Tree-REX [23], qui explore l'espace de recherche de manière similaire à totSAT, mais dont la traduction en logique propositionnelle est spécifiquement conçue pour la résolution SAT incrémentale, ce qui améliore les performances et réduit la taille des encodages. À la suite de ces travaux, Schreiber et al. ont introduit Lilotane [21], un successeur de Tree-REX et le premier planificateur TOHTN basé sur SAT utilisant une approche lifted. Lilotane évite le processus de mise à plat coûteux en utilisant une instanciation différée des tâches et méthodes, permettant des arguments libres lorsque nécessaire. Cela conduit à des formules SAT considérablement plus petites que les encodages précédents.

10 Conclusion

Dans cet article, nous avons présenté une nouvelle approche basée sur SAT pour résoudre les problèmes de planification TOHTN. Notre approche utilise un solveur SAT à la fois pour rechercher des solutions et pour guider l'exploration de l'espace de recherche. Plus précisément, nous avons montré comment l'encodage d'un problème relaxé peut permettre de dégager une heuristique grâce à un solveur SAT, qui peut ensuite être utilisée pour identifier des zones prometteuses dans l'espace de recherche. Nous avons démontré que notre planificateur, qui utilise cette heuristique pour explorer l'espace de manière gloutonne, surpasse les autres planificateurs TOHTN basés sur SAT à l'état de l'art en termes de temps d'exécution et de qualité des plans. Ce travail ouvre la voie à l'intégration d'informations heuristiques dans la planification TOHTN basée sur SAT.

Dans les travaux futurs, nous prévoyons d'explorer des méthodes allant au-delà de notre stratégie de recherche actuelle, qui repose sur une approche gloutonne. Plus précisément, nous souhaitons étudier si l'identification de plusieurs zones prometteuses dans l'espace de recherche à l'aide de notre heuristique, suivie d'un classement et d'une priorisation à l'aide d'heuristiques HTN classiques [13, 14, 15], pourrait améliorer encore davantage les performances.

11 Remerciements

Nous remercions les relecteurs pour leurs commentaires et suggestions. Ce travail a été partiellement soutenu par MIAI@Grenoble Alpes, (ANR-19-P3IA-0003).

Références

[1] Ron Alford, Gregor Behnke, Daniel Höller, Pascal Bercher, Susanne Biundo, and David Aha. Bound to plan : Exploiting classical heuristics via automatic translations of tail-recursive HTN problems. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 26, pages 20–28, 2016.

[2] Ronald Alford, Ugur Kuter, and Dana S Nau. Translating HTNs to PDDL : A Small Amount of Domain Knowledge Can Go a Long Way. In *IJCAI*, volume 9, pages 1629–1634, 2009.

[3] Gilles Audemard and Laurent Simon. On the glucose SAT solver. *International Journal on Artificial Intelligence Tools*, 27(01) :1840001, 2018.

[4] Gregor Behnke. Block compression and invariant pruning for SAT-based totally-ordered HTN planning. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 31, pages 25–35, 2021.

[5] Gregor Behnke, Daniel Höller, and Pascal Bercher, editors. *Proceedings of the 10th International Planning Competition : Planner and Domain Abstracts – Hierarchical Task Network (HTN) Planning Track (IPC 2020)*, 2021.

[6] Gregor Behnke, Daniel Höller, and Susanne Biundo. totSAT-Totally-ordered hierarchical planning through SAT. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[7] Gregor Behnke, Daniel Höller, and Susanne Biundo. Bringing order to chaos–A compact representation of partial order in SAT-based HTN planning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 7520–7529, 2019.

[8] Gregor Behnke, Daniel Höller, and Susanne Biundo. Finding Optimal Solutions in HTN Planning-A SAT-based Approach. In *IJCAI*, pages 5500–5508, 2019.

[9] Gregor Behnke, Florian Pollitt, Daniel Höller, Pascal Bercher, and Ron Alford. Making translations to classical planning competitive with other HTN planners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 9687–9697, 2022.

[10] Kutluhan Erol, James A Hendler, and Dana S Nau. UMCP : A Sound and Complete Procedure for Hierarchical Task-network Planning. In *Aips*, volume 94, pages 249–254, 1994.

[11] Daniel Fišer. Lifted fact-alternating mutex groups and pruned grounding of classical planning problems. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 9835–9842, 2020.

[12] Thomas Geier and Pascal Bercher. On the decidability of HTN planning with task insertion. In *IJCAI Proceedings-International Joint Conference on Artificial Intelligence*, volume 22, page 1955, 2011.

[13] Daniel Höller, Pascal Bercher, Gregor Behnke, and Susanne Biundo. A generic method to guide HTN progression search with classical heuristics. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 28, pages 114–122, 2018.

[14] Daniel Höller, Pascal Bercher, Gregor Behnke, and Susanne Biundo. On Guiding Search in HTN Planning with Classical Planning Heuristics. In *IJCAI*, pages 6171–6175, 2019.

[15] Daniel Höller, Pascal Bercher, Gregor Behnke, and Susanne Biundo. HTN planning as heuristic progression search. *Journal of Artificial Intelligence Research*, 67 :835–880, 2020.

[16] Maurício C Magnaguagno, Felipe Rech Meneguzzi, and Lavindra De Silva. HyperTensioN : A three-stage compiler for planning. In *Proceedings of the 30th International Conference on Automated Planning and Scheduling (ICAPS), 2020, França.*, 2020.

[17] Amol Dattatraya Mali and Subbarao Kambhampati. Encoding HTN Planning in Propositional Logic. In *AIPS*, pages 190–198, 1998.

[18] Van-Hau Nguyen, Van-Quyet Nguyen, Kyungbaek Kim, and Pedro Barahona. Empirical Study on SAT-Encodings of the At-Most-One Constraint. In *The 9th International Conference on Smart Media and Applications*, pages 470–475, 2020.

[19] Conny Olz and Pascal Bercher. Can They Come Together? A Computational Complexity Analysis of Conjunctive Possible Effects of Compound HTN Planning Tasks. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 33, pages 314–323, 2023.

[20] Conny Olz, Susanne Biundo, and Pascal Bercher. Revealing hidden preconditions and effects of compound HTN planning tasks–a complexity analysis. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 11903–11912, 2021.

[21] Dominik Schreiber. Lilotane : A lifted SAT-based approach to hierarchical planning. *Journal of artificial intelligence research*, 70 :1117–1181, 2021.

[22] Dominik Schreiber, Damien Pellier, Humbert Fiorino, and Tomáš Balyo. Efficient SAT encodings for hierarchical planning. In *11th International Conference on Agents and Artificial Intelligence (ICAART 2019)*, 2019.

[23] Dominik Schreiber, Damien Pellier, Humbert Fiorino, et al. Tree-REX : SAT-based tree exploration for efficient and high-quality HTN planning. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 29, pages 382–390, 2019.

[24] Ayal Taitler, Ron Alford, Joan Espasa, Gregor Behnke, Daniel Fišer, Michael Gimelfarb, Florian Pommerening, Scott Sanner, Enrico Scala, Dominik Schreiber, et al. The 2023 International Planning Competition, 2024.