Optiplan : Encodage CSP pour planification HTN-POCL optimale

Oleksandr Firsov, Humbert Fiorino, Damien Pellier

Université Grenoble Alpes - LIG

1er mars 2025

Résumé

La planification HTN est une approche efficace pour résoudre des problèmes complexes du monde réel. Un défi croissant est d'assurer la flexibilité des plans, car une certaine liberté d'exécution les rend plus résistants aux changements. Cette flexibilité est obtenue en combinant HTN avec POCL. Dans cet article, nous présentons Optiplan, une méthode d'encodage novatrice qui modélise ces problèmes sous forme de CSP et, pour la première fois, aborde l'optimisation des solutions.

Mots-clés

Planification, IA, partiellement ordonné, HTN, encodage, CSP

Abstract

HTN planning is an effective approach for solving complex real-world problems. A growing need in this context is for plan flexibility, as allowing some degree of freedom in task execution makes plans more resilient to changes. This flexibility is achieved by combining HTN with POCL planning. In this paper, we introduce Optiplan, a novel encoding method that models these planning problems as CSPs and also, for the first time, tackles the question of solution optimization.

Keywords

Planning, AI, partially ordered, HTN, encoding, CSP

1 Introduction

Au cours des dernières années, les approches de planification hiérarchique ont suscité un intérêt de recherche considérable, les réseaux de tâches hiérarchiques (Hierarchical Task Networks, HTNs) étant largement adoptés dans divers domaines pratiques[5]. La popularité des HTNs repose sur leur structure hiérarchique, qui leur permet de capturer des informations riches, spécifiques à un domaine, et d'utiliser ces connaissances pour guider la planification de manière efficace.

Au coeur de la planification HTN est l'idée de décomposer les problèmes complexes en une hiérarchie de tâches. Ce processus commence par des tâches abstraites de haut niveau, qui sont progressivement affinées en sous-tâches plus concrètes à l'aide de "recettes" préétablies, jusqu'à atteindre des actions primitives exécutables. Cette approche présente deux avantages majeurs : d'une part, elle améliore la scalabilité des problèmes, et d'autre part, elle offre aux utilisateurs un meilleur contrôle sur le processus de planification.

La majorité des planificateurs HTN se concentrent sur la génération de plans de solution totalement ordonnés en utilisant diverses techniques, telles que l'encodage SAT [3, 20, 19], l'encodage CSP [22, 21], la recherche heuristique [13, 6] ou encore la traduction du problème en planification classique STRIPS [1, 4, 9].

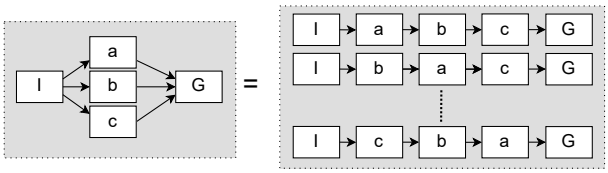

FIGURE 1 – À gauche, un plan partiellement ordonné. À droite, 3 des 6 plans totalement ordonnés déductibles.

Bien que ces approches se soient révélées efficaces [19], les plans obtenus suivent un ordre d'actions fixe, ne laissant aucune marge d'ajustement pendant l'exécution. En d'autres termes, ils manquent de flexibilité, c'est-à-dire de la capacité à s'adapter à des circonstances changeantes (voir Fig. 1). Cela peut constituer une limitation majeure dans des applications où un certain degré de liberté est essentiel pour gérer l'incertitude environnementale et les risques potentiels, comme par exemple dans la planification de missions pour robots [16] ou la planification industrielle [23].

Plusieurs approches ont été proposées pour introduire de la flexibilité dans la planification hiérarchique, comme le documentent [15, 6]. Ces approches combinent la structure hiérarchique avec le concept de liens causaux à ordre partiel (Partial Order Causal Links, POCL) [17, 18], qui suit le principe du "moindre engagement", c'est-à-dire que le planificateur ne spécifie l'ordre des tâches que lorsque cela est nécessaire. Pour faciliter la lecture, nous ferons référence à la planification hiérarchique partiellement ordonnée sous le nom de *planification HTN-POCL*.

Dans cet article, nous présentons Optiplan - une nouvelle méthode d'encodage capable de produire des plans partiellement ordonnés avec un chemin critique optimal (c'est-à-dire des plans où la plus longue chaîne d'activités dépendantes est minimisée). Il exploite des techniques CSP et s'appuie sur les avancées récentes en encodages pour la pla-

nification totalement ordonnée. À notre connaissance, Optiplan est la première approche capable de générer des plans HTN-POCL optimaux et la première à utiliser un CSP, ou toute autre forme d'encodage, dans ce contexte.

L'article est structuré comme suit. Tout d'abord, nous introduisons les concepts associés à la planification HTN-POCL. Ensuite, nous présentons en détail notre approche, en abordant la structure de données utilisée pour représenter l'espace des plans, la méthode d'encodage employée ainsi que la méthodologie d'extraction d'un plan de solution. Par la suite, nous réalisons une analyse comparative d'Optiplan par rapport à d'autres planificateurs utilisant la planification HTN partiellement ordonnée, en évaluant la performance en termes de qualité des plans (i.e., la longueur du chemin critique du plan généré) et d'agilité du planificateur (i.e., le temps nécessaire pour produire un plan). Enfin, nous discutons des travaux connexes.

2 Planification HTN partiellement ordonnée

Cette section introduit les fondements de la planification HTN-POCL.

2.1 Actions, Méthodes et Tâches

Une *proposition* logique est un objet atomique, et un *état* s est un ensemble cohérent de propositions. Une action est un triplet *a = (name(a), pre(a), eff(a))* où *name(a)* est l'identifiant de a, *pre(a)* définit les *préconditions* qui doivent être satisfaites pour appliquer l'action, et *eff(a)* définit les *effets* qui s'appliquent à l'état après l'exécution de a : $\mathit{eff}(a) = (\mathit{eff}^+(a), \mathit{eff}^-(a))$ avec $\mathit{eff}^+(a) \cap \mathit{eff}^-(a) = \emptyset$.

Une action a est *applicable* à un état s si $pre(a) \subseteq s$. L'état résultant s' de l'application de a à un état s est défini comme suit : $s' = \gamma(s, a) = (s \setminus \mathit{eff}^-(a)) \cup \mathit{eff}^+(a)$.

L'application d'une séquence d'actions est définie récursivement par $\gamma(s, []) = s$ et $\gamma(s, [a_0, a_1, \ldots, a_n]) = \gamma(\gamma(s, a_0), [a_1, \ldots, a_n])$.

Un *réseau de tâches* w est un couple $(T, \prec)$ où T est une séquence d'identifiants de tâches $[\ldots, t_i, \ldots]$, et $\prec$ est un ensemble de contraintes d'ordre sur T : t_i *précède* t_j si et seulement si $(t_i < t_j) \in \prec$.

Il est important de noter que la séquence de tâches T est totalement arbitraire, n'a aucun lien direct avec $\prec$, et est simplement utilisée comme une commodité pour construire la structure de données d'encodage d'Optiplan, à savoir l'*arbre de décomposition HTN-POCL* (voir section suivante).

Une *méthode* $m = (name(m), (T, \prec))$ est un couple où *name(m)* est l'identifiant de m, et $(T, \prec)$ est un réseau de tâches décrivant comment une méthode est accomplie.

Une action a *peut accomplir* une tâche t si *name(a)* $= t$, et une méthode m *peut accomplir* une tâche t si *name(m)* $= t$. Il est important de noter qu'une même tâche t peut être accomplie par différentes méthodes ou actions.

Nous notons $R(t)$ l'ensemble de toutes les actions et/ou méthodes qui peuvent accomplir t. Nous appelons ces actions et/ou méthodes les *accomplisseurs* de t. [1]

La manière dont ces accomplisseurs sont sélectionnés pour effectivement accomplir une tâche donnée sera expliquée plus tard dans cette section. Nous noterons la *taille* d'une méthode m (c'est-à-dire le nombre de tâches dans son réseau de tâches) par $\sharp m$. La taille de toute action est égale à 1.

2.2 Liens causaux, Plans partiels, Menaces et Défauts

Un *lien causal* est un triplet (a, p, a'), où p est une proposition, a est une action produisant p (i.e., $p \subseteq \mathit{eff}^+(a)$), et a' est une action ayant p parmi ses préconditions. Les liens causaux sont notés $a \xrightarrow{p} a'$, et ils indiquent les dépendances producteur/consommateur entre les actions d'un plan. Les liens causaux imposent des contraintes sur l'ordre des actions, puisque $a \xrightarrow{p} a'$ requiert que a précède a'.

Un *plan partiel* π est un triplet (w, α, C) où :

— w est un réseau de tâches $(T, \prec)$ qui définit les tâches et les contraintes d'ordre de π,

— α est une correspondance **partielle** de $t \in T$ vers une action ou une méthode accomplissant t. Si t est accompli par une action, alors t est considéré comme *primitif*, sinon il est *non primitif* (i.e., que soit $\alpha(t) = m$, soit $\alpha(t)$ est indéfini),

— C est un ensemble de liens causaux $a \xrightarrow{p} a'$, où a et a' sont deux actions accomplissant respectivement t et t' dans T.

Une *menace* sur un lien causal $a \xrightarrow{p} a'$ est une action a'' dans π telle que $p \in \mathit{eff}^-(a'')$ (i.e., que a'' détruit p), et où $\prec$ n'inclut pas les contraintes d'ordre $(a'' < a)$ ou $(a' < a'')$. En d'autres termes, a'' constitue une menace s'il est possible qu'il se produise entre a et a', rendant ainsi l'exécution de a' impossible.

Un *objectif ouvert* (open goal) dans π est une précondition d'une action a qui n'est soutenue par aucun lien causal. Plus formellement, $p \in pre(a)$ est un objectif ouvert s'il n'existe pas d'action a' dans le plan partiel telle que $a' \xrightarrow{p} a \in C$.

Un *défaut* dans π est soit une tâche non primitive, soit une menace, soit un objectif ouvert.

Une solution à un problème de planification est un plan partiel *sans défaut*. Ce plan peut également être représenté sous forme d'un graphe orienté acyclique (DAG) avec l'action initiale I comme source et l'action objectif G comme sommet terminal (voir Figure 1). Dans ce DAG, le chemin critique correspond au plus court chemin de I à G.

2.3 Problème de planification

Un *problème de planification HTN-POCL* P est défini comme un tuple $(\mathcal{L}, \mathcal{T}, \mathcal{A}, \mathcal{M}, \pi_0, s_0, g)$ où :

— $\mathcal{L}$ est un ensemble de propositions,

1. Sans perte de généralité, nous ne considérons pas ici les méthodes m ayant des préconditions $pre(m)$, car elles sont équivalentes à des méthodes telles qu'il existe une tâche $t \in T_m$ qui précède toutes les autres tâches, où une action a accomplit t, et où $\mathit{pre}(m) = \mathit{pre}(a)$, $\mathit{eff}^+(a) \cup \mathit{eff}^-(a) = \emptyset$.

Algorithm 1 SOLVE(P)

Soit f un défaut dans π
if f est une tâche non primitive t **then**
  $\pi' = \text{DECO}(\pi, t)$
  return SOLVE(P)
else if f est un objectif ouvert p dans l'action a_j de π **then**
  $a_i =$ *choisirProducteur*(a_j, p) `// arbitraire`
  if $a_i = \emptyset$ **then**
    return FAILURE
  else
    $\pi' = (w, \alpha, C \cup (a_i \xrightarrow{p} a_j))$
    $P' = (\mathcal{L}, \mathcal{T}, \mathcal{A}, \mathcal{M}, \pi', s_0, g)$
    return SOLVE(P')
  end if
else
  return π
end if

— $\mathcal{T}$ est un ensemble de noms de tâches,
— $\mathcal{A}$ est un ensemble d'actions,
— $\mathcal{M}$ est l'ensemble des méthodes,
— π_0 est le plan initial à décomposer,
— s_0 et g sont deux sous-ensembles de $\mathcal{L}$ représentant respectivement l'état initial du problème et l'objectif à atteindre, s'il existe.

Afin de rester cohérent avec la littérature POCL, le plan initial π_0 est défini comme $(w_0, \emptyset, \emptyset)$, où w_0 contient les tâches abstraites initiales ainsi que deux tâches spéciales, t_0 et t_∞, qui représentent respectivement l'état initial et l'état objectif.
La tâche t_0 est accomplie par une action a_0, où *pre*$(a_0) = \emptyset$ et *eff*$(a_0) = s_0$, et la tâche t_∞ est accomplie par une action a_∞, où *pre*$(a_\infty) = g$ et *eff*$(a_\infty) = \emptyset$. Par conséquent, t_0 et t_∞ sont respectivement ordonnées comme la première et la dernière tâche de w_0.

2.4 Procédure de résolution et raffinement du plan

Un problème de planification P est résolu par une procédure récursive SOLVE(P) (voir Algo. 1), qui corrige les défauts sans introduire de menaces. Alors qu'un objectif ouvert est accompli en établissant un lien causal, un défaut introduit par une tâche abstraite t_i est résolu par un raffinement ou une concrétisation du plan partiel par rapport à t_i. Cette procédure de raffinement est appelée DECO(π, t_i) (voir Algo. 2), car elle applique une décomposition de la méthode associée à t_i sur π.
Lorsque DECO(P) applique une méthode, elle introduit des sous-tâches dans le réseau de tâches. La fonction *integrerOrdonnancement* indique que les contraintes d'ordre des sous-tâches doivent être incorporées dans les contraintes du réseau de tâches. De plus, si la tâche parente possède des relations d'ordre, celles-ci sont héritées par les sous-tâches. Ainsi, DECO(P) affine les plans partiels en leurs versions plus concrètes. Cependant, cette procédure ne modifie pas

Algorithm 2 DECO(π, t_i)

Soit $r \in R(t_i)$
if r est une action a **then**
  if r est une **menace then**
    return FAILURE
  else
    return $\pi' = (w, \alpha \cup \alpha(t_i) = a, C)$
  end if
else if r est une méthode m **then**
  $\alpha' = \alpha \cup \alpha(t_i) = m$
  $T' = T \cup T^m$
  $\prec' =$ *integrerOrdonnancement*$(\prec, \prec^m)$
  $w' = (T', \prec')$
  return $\pi' = (w', \alpha', C)$
end if

les liens causaux, ce qui signifie que bien que les plans obtenus soient exempts de menaces, ils ne sont pas encore parfaits. C'est SOLVE(P) qui introduit les liens causaux, aboutissant ainsi à une solution. Cette procédure est définie récursivement comme suit :

3 L'approche Optiplan

Dans cette section, nous décrivons les principes d'encodage de SOLVE(P) en un problème de satisfaction de contraintes (Constraint Satisfaction Problem, CSP).
L'espace de recherche d'Optiplan est défini par un *arbre de décomposition HTN-POCL* (arbre HiPOD), qui représente la structure hiérarchique des décompositions possibles d'un plan. L'arbre HiPOD est constitué de couches qui organisent les actions et les méthodes sous une structure arborescente, chaque couche représentant un niveau de décomposition. Cette approche est inspirée de [20], bien que dans notre cas, les éléments de chaque couche ne soient pas ordonnés, et nous explicitons les préconditions ainsi que les liens causaux.
La procédure d'encodage suit une recherche *en largeur* sur l'ensemble des décompositions possibles, en partant des noeuds racines correspondant aux tâches initiales. Elle est conçue comme un processus *incrémental*, ce qui signifie que HiPOD est construit jusqu'à une certaine profondeur k à chaque étape de décomposition. L'arbre est ensuite soumis à un solveur CSP. Si le solveur ne retourne pas de solution à cette profondeur, le niveau de HiPOD est incrémenté et la procédure est répétée jusqu'à ce qu'une solution soit trouvée ou qu'une limite arbitraire $k_{\max}$ soit atteinte, suivant [3].
Ainsi, Optiplan a la propriété suivante : étant donné un problème de planification HTN-POCL P et une profondeur k, il existe une correspondance biunivoque entre l'ensemble des solutions de P de profondeur $\leq k$ et l'ensemble des solutions du CSP. En d'autres termes, SOLVE(P) et notre encodage sont *congruents*. À partir d'une solution du CSP, si elle existe, la correspondance permet d'obtenir une solution à P. Si aucune solution n'existe, alors aucun plan de profondeur $\leq k$ n'existe.

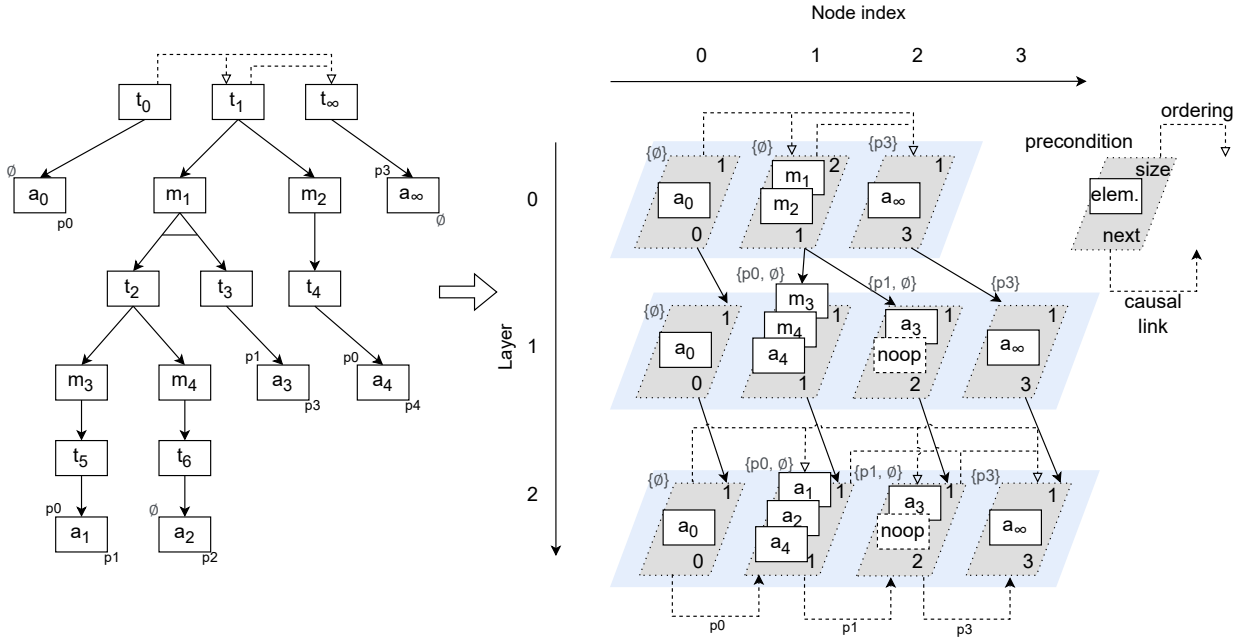

FIGURE 2 – À gauche, une décomposition représentée sous forme d'arbre ET/OU, où t représente une tâche, a une action et m une méthode. p représente une proposition : si elle est placée au-dessus d'un noeud, elle correspond à une précondition ; sinon (en dessous d'un noeud), elle représente un effet. Les flèches sortantes des tâches représentent les relations "est accompli par", et les flèches sortantes des méthodes indiquent les décompositions en sous-tâches. À droite, l'arbre HiPOD correspondant. Les éléments empilés forment $cell(i, j)$ où i correspond à la couche L_i, et j correspond à l'indice de la cellule dans le tableau L_i.

Cela garantit qu'Optiplan est à la fois *correct* et *complet*. La procédure est correcte, car chaque solution retournée par le solveur CSP est un plan valide qui satisfait toutes les contraintes du problème de planification HTN-POCL. Elle est complète, car si une solution de profondeur $\leq k$ existe, le solveur CSP est assuré de la trouver, garantissant qu'aucune solution valide n'est omise.

Dans la suite, nous détaillons le processus de construction de HiPOD et fournissons les règles d'encodage du CSP.

3.1 Arbre de décomposition HTN-POCL

L'arbre HiPOD organise la décomposition hiérarchique d'un problème de planification en plusieurs niveaux, représentés comme une séquence de couches hiérarchiques $[L_0, \ldots, L_i, \ldots, L_n]$. Chaque *couche hiérarchique* est un tableau de cellules, où chaque cellule contient un ensemble d'actions et de méthodes, désigné comme le domaine de la cellule. Nous notons la j-ième cellule de la couche L_i sous la forme $cell(i, j)$. Les cellules peuvent également contenir *noop*, une action sans effets ni préconditions (voir Fig. 2).

Chaque cellule possède une liste de préconditions associées. Comme les actions au sein d'une cellule peuvent avoir des préconditions différentes, les préconditions des cellules sont variables et leur domaine est défini par $pre_i(a), a \in cell$, où i est un indice dans la liste des préconditions. Un exemple de préconditions de cellules est illustré en Fig. 3.

Chaque précondition de cellule doit être soutenue par un lien causal. Par conséquent, chaque couche hiérarchique contient une liste de *liens causaux*, qui capture les dépendances causales potentielles entre les cellules. De même, chaque couche dispose d'une liste de *contraintes d'ordre*, qui définit l'ordre partiel des actions dans la couche. Le niveau i d'une couche hiérarchique L_i indique le nombre de décompositions qui doivent être appliquées depuis le réseau de tâches initial w_0 pour atteindre une action dans les cellules de L_i.

La première couche L_0 est construite directement à partir de w_0 comme suit : pour chaque tâche de w_0, nous créons une cellule correspondante dans L_0, où l'ordre des cellules est arbitraire. Le domaine de chaque cellule est rempli avec les actions et les méthodes capables d'accomplir la tâche correspondante. Par exemple, dans la Figure 2, la première cellule de L_0 est $(cell(0, 0) \in a_0)$ et elle correspond à t_0. Pour la tâche t_1 de w_0, nous avons $cell(0, 1) \in m_1, m_2$. Enfin, $cell(0, 2) \in a_\infty$ est obtenu à partir de t_∞.

Le réseau de tâches initial peut imposer certaines contraintes d'ordre entre les tâches. Ces contraintes sont enregistrées dans la liste des contraintes d'ordre pour L_0.

Chaque couche suivante L_{i+1} est construite de manière incrémentale à partir des cellules de L_i. Chaque $cell(i, j)$ dans L_i est un parent d'un certain nombre de cellules dans L_{i+1}. Le nombre de cellules enfants est déterminé par le plus grand nombre de sous-tâches dans toute méthode pré-

sente dans la cellule. Nous définissons ce nombre comme $size(i,j)$, qui est calculé comme suit :

$$size(i,j) = max\sharp e | \forall e \in cell(i,j) \quad (1)$$

Les enfants de $cell(i,j)$ sont indexés dans L_{i+1} en calculant leur position relative au parent. Nous notons la position du premier enfant de $cell(i,j)$ sous la forme $next(i,j)$, qui est calculée récursivement comme suit :

$$next(i,j) = \begin{cases} 0 & \text{si } j = 0 \\ next(i,j-1) + size(i,j-1) & \text{sinon} \end{cases} \quad (2)$$

Ainsi, les enfants de $cell(i,j)$ sont les cellules positionnées dans l'intervalle $[next(i,j), next(i,j) + size(i,j) - 1]$.
Par exemple, dans Fig. 2, comme $size(0,1) = 2$, $cell(0,1)$ génère deux enfants dans L_1, correspondant à $cell(1,1)$ et $cell(1,2)$.
Le domaine de chaque cellule enfant est dérivé récursivement du domaine de sa cellule parente dans L_i :

— si e est une action a, alors a est ajoutée à $cell(i+1, next(i,j))$,

— si e est une méthode m avec des sous-tâches $[t_0, \ldots, t_k, \ldots, t_n]$, alors pour chaque k, les accomplisseurs $R(t_k)$ sont placés dans $cell(i+1, next(i,j)+k)$.

3.1.1 Analyse de l'accessibilité

Avant d'encoder un problème en CSP, Optiplan effectue d'abord une *analyse de l'accessibilité* sur l'arbre à la profondeur actuelle k. Cette analyse permet d'éviter des tentatives de résolution inutiles lorsque HiPOD n'a pas été suffisamment développé pour contenir une solution.
D'après la définition de SOLVE(P), nous savons qu'une couche entraînera un FAILURE si l'une des conditions suivantes est remplie :

— Il existe une cellule $cell(i,j) \in L_i$ qui ne contient aucune action dans son domaine. Cela implique que tous les plans auront un défaut de tâche non primitive.

— Pour chaque précondition de chaque action dans $cell(i,j)$, il n'existe aucun lien causal dans la liste des liens causaux de L_i.

La première condition est triviale à vérifier. La seconde condition peut être vérifiée en temps polynomial : pour chaque précondition p de chaque action a dans $cell(i,j)$, nous vérifions s'il existe une autre cellule $cell(i,k) \in L_i$ (avec $j \neq k$) contenant une action b telle que $p \in eff^+(b)$. En d'autres termes, nous vérifions si les préconditions de chaque action dans $cell(i,j)$ peuvent être satisfaites par les effets d'une action présente dans une autre cellule au même niveau de profondeur. Si aucune cellule ne permet de satisfaire ces préconditions pour toutes les valeurs du domaine de $cell(i,j)$, alors quelle que soit l'action choisie, la condition d'échec sera remplie.
Nous pouvons maintenant passer à l'explication de la synthèse d'un CSP à partir de l'arbre HiPOD.

3.2 Encodage CSP

La satisfaction de contraintes est un paradigme général et puissant pour la résolution de problèmes [10], qui peut être formalisé comme un triplet (X, D, C), où :

— $X = \langle x_1, ..., x_n \rangle$ est un ensemble de variables,

— $D = \langle D_1, ..., D_n \rangle$ est un ensemble de domaines associés aux ces variables $x_i \in D_i$,

— $C = \langle c_1, ..., c_m \rangle$ est un ensemble de contraintes sur les variables. Une contrainte d'arité k restreint les valeurs d'un tuple $(x_1, ..., x_k) \in X$. Elle peut être notée soit sous la forme $((x_1, ..., x_k), \{(v_1, ..., v_k) \in D_1, ..., D_k\})$, soit $\{(x_1 = v_1, ..., x_k = v_k) | (v_1, ..., v_k) \in D_1, ..., D_k\}$. À noter que dans l'article, nous remplaçons parfois des valeurs par _, par exemple $(x_1 = a, x_2 = \_)$, ce qui signifie que toute valeur du domaine de cette variable est valide.

Une *solution* à un CSP est un tuple de valeurs $T = (v_1, ..., v_n)$ tel que $v_i \in D_i$, et que l'affectation des variables $x_i = v_i$ satisfait la conjonction de toutes les contraintes de C.

3.2.1 Variables CSP

Dans l'approche Optiplan, nous avons besoin de trois types de variables de décision correspondant 1) au choix d'un élément de cellule, 2) à l'établissement des liens causaux et 3) à la sélection des contraintes d'ordre :

— $v_{i,j}$. Pour chaque $cell(i,j)$, il existe une variable $v_{i,j}$ dont le domaine correspond au domaine de la cellule. Par conséquent, la valeur de $v_{i,j}$ représente l'élément choisi dans la cellule.

— $o_{i,j,k}$. Cette variable indique une relation d'ordre entre $cell(i,j)$ et $cell(i,k)$. Son domaine est $\{<, >, \emptyset\}$.

— $\lambda^p_{i,j}$. Cette variable indique quelle cellule de L_i soutient la précondition p de $cell(i,j)$. Étant donné que les préconditions exactes de $cell(i,j)$ ne sont pas connues avant la sélection d'un élément dans la cellule, p représente un indice de précondition plutôt qu'un fait spécifique. Par exemple, dans la Fig. 3, la précondition 0 de $cell(0,0)$ est $x1$ si $cell(0,0) = a_1$, $\emptyset$ si $cell(0,0) = a_2$, et $x3$ si $cell(0,0) = a_3$.

Sur ces variables, nous appliquons les contraintes suivantes :

C1. Les valeurs des cellules parentes et enfants doivent être cohérentes avec les décompositions. Soit $cell(i+1, l), l \in [next(i,j), next(i,j) + size(i,j)]$ un enfant de $cell(i,j)$. Chaque élément e dans la cellule introduit une contrainte définie comme suit :

— Si $e \in \mathcal{M}$ avec des sous-tâches $[t_0, ..., t_k, ..., t_n]$, alors ajouter $(v_{i,j} = e, v_{i+1,l} = e'), |, e' \in R(t_k)$.

— Si $e \in \mathcal{M}$ mais $k > |w|$, alors $(v_{i,j} = e, v_{i+1,l} = noop)$.

— Si $e \in \mathcal{A}$ et $k = 0$, alors $(v_{i,j} = e, v_{i+1,l} = e)$.

— Si $e \in \mathcal{A}$ et $k > 0$, alors $(v_{i,j} = e, v_{i+1,l} = noop)$.

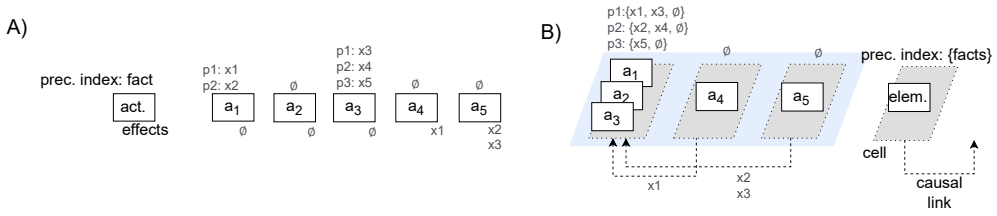

FIGURE 3 – À gauche, quelques actions avec leurs effets et préconditions. À droite, une couche hiérarchique.

Dans Fig. 2, $cell(0,1)$ génère deux contraintes :

— $(m1, m3), (m1, m4), (m2, a4)$ sur $(v_{0,1}, v_{1,1})$

— $(m1, a3), (m2, noop)$ sur $(v_{0,1}, v_{1,2})$

C2. Chaque précondition de $cell(i,j)$ est supportée par une autre cellule. Cette contrainte binaire s'applique aux couples $(v_{i,j}, \lambda^p_{i,j})$ avec $j \in [0, |L_i| - 1]$, où p désigne la p-ième précondition de $cell(i,j)$. Pour chaque e dans la cellule, les contraintes suivantes sont ajoutées :

— Si $e \in \mathcal{M}$, alors ajouter $(v_{i,j} = e, \lambda^k_{i,j} = \emptyset)$.

— Si $e \in \mathcal{A}$ dont la p-ième précondition est x, alors ajouter $(v_{i,j} = e, \lambda^p_{i,j} = l), |, \exists a \in D(v_{i,l}), x \in \mathit{eff}^+(a))$.

— Si $e \in \mathcal{A}$ et qu'il a moins de p préconditions, alors ajouter $(v_{i,j} = e, \lambda^p_{i,j} = \emptyset)$.

Dans Fig. 3, $cell(0,0)$ génère une contrainte par precondition :

— $(a, q), (b, q), (c, f)$ sur $(v_{i,j}, \lambda^0_{i,j})$

— $(a, f), (b, \emptyset)$ sur $(v_{i,j}, \lambda^1_{i,j})$

— $(a, \emptyset), (b, \emptyset)$ sur $(v_{i,j}, \lambda^2_{i,j})$

On remarque que dans cet exemple, $v_{i,j} = c$ est impossible, car la deuxième contrainte restreint le domaine de $v_{i,j}$ à a, b.

C3. Lorsqu'une cellule $cell(i,l)$ supporte une précondition de $cell(i,j)$, l'action sélectionnée doit la produire. Il s'agit d'une contrainte ternaire sur les triplets $(v_{i,j}, \lambda^p_{i,j}, v_{i,l})$. Ses valeurs possibles sont :

— $(v_{i,j} = e, \lambda^k_{i,j} = l, v_{i,l} = e')$, tel que $pre_p(e) \in \mathit{eff}^+(e')$

— $(v_{i,j} = \_, \lambda^k_{i,j} \neq l, v_{i,l} = \_)$

Dans Fig. 3, par exemple, nous avons deux contraintes pour la première précondition de $cell(i,j)$:

— $(a, q, d), (b, q, d), (\_, f, \_)$ sur $(v_{i,j}, \lambda^0_{i,j}, v_{i,q})$

— $(c, f, e), (\_, q, \_)$ sur $(v_{i,j}, \lambda^0_{i,j}, v_{i,f})$

C4. w_0 impose des relations d'ordre entre les cellules de L_0. Cette contrainte unaire est introduite pour chaque relation d'ordre $t_j < t_k$ dans $\prec_0$: $(o_{0,j,k}, <)$.

C5. Les relations d'ordre sont transitives. Cette contrainte capture le comportement suivant : si $cell(i,j) < cell(i,k)$ et $cell(i,k) < cell(i,q)$ alors $cell(i,j) < cell(i,q)$. Cette transitivité est assurée par une contrainte ternaire sur chaque triplet $(o_{i,j,k}, o_{i,k,q}, o_{i,j,q})$, avec les valeurs autorisées étant $(x, y, z), |, x, y, z \in <, >, \emptyset, x = \emptyset \vee x \neq y \vee y = z$.

C6-7. La cellule d'état initial est toujours la première, et la cellule d'état objectif - la dernière. Pour chaque $j \in [1, |L_i|]$, nous définissons une contrainte unaire $(o_{i,0,j} = ; <)$. Et pour chaque $j \in [0, |L_i| - 2]$, nous définissons $(o_{i,j,|L_i|-1} = ; <)$.

C8. Les enfants héritent des relations d'ordre de leurs parents. Cette contrainte binaire s'applique aux couples $(o_{i,j,k}, o_{i+1,g,q})$ tels que $g \in [next(i,j), next(i,j) + size(i,j)]$ et $q \in [next(i,k), next(i,k) + size(i,k)]$. Ses valeurs autorisées sont $(o_{i,j,k} = x, o_{i+1,g,q} = y), |, x, y \in <, >, \emptyset, x = \emptyset \vee x = y$.

C9. Les méthodes de L_i peuvent introduire des ordres dans L_{i+1}. Cette contrainte s'applique aux couples $(v_{i,j}, o_{i+1,next(i,j)+g,next(i,j)+k})$ tels que $g, k \in [0, size(i,j)]$. Chaque $m \in v_i(j)$ introduit des valeurs comme suit :

— $(v_{i,j} = m, o_{i+1,\ldots} = \circ), |, (t_g \circ t_k) \in \prec_m$

— si m ne définit pas d'ordre sur les positions g, k, alors $(v_{i,j} = m, o_{i+1,\ldots} = \_)$, c'est-à-dire que tout ordre est autorisé.

C10. Un lien causal implique une relation d'ordre. Cette contrainte binaire s'applique aux couples $(\lambda^p_{i,j}, o_{i,l,j})$. Les valeurs autorisées sont $(\lambda^p_{i,j} = l, o_{i,l,j} = ; <)$, ainsi que $(\lambda^p_{i,j} \neq l, o_{i,l,j} = \_)$.

C11. Les liens causaux ne doivent pas être menacés. Cette contrainte s'applique aux tuples $(\lambda^p_{i,j}, v_{i,j}, v_{i,q}, o_{i,j,q}, o_{i,l,q})$ pour éviter que a' ne détruise la précondition de a.

C12. Seules les actions peuvent être sélectionnées dans la dernière couche L_n. Chaque cellule $j \in [0, |L_n| - 1]$ produit une contrainte unaire $v_{n,j} = a, |, a \in \mathcal{A}$.

3.3 Extraction du plan

Une fois que le problème P avec une borne k a été encodé en CSP et résolu, l'extraction d'un plan partiellement ordonné implique deux étapes : 1) la construction du réseau de tâches w, et 2) l'identification de l'ensemble des liens causaux C. Ces deux éléments peuvent être déduits à partir de l'affectation des variables dans la couche finale L_n. Les tâches de w sont directement obtenues à partir des valeurs attribuées aux variables v. Les contraintes d'ordre entre ces tâches sont ensuite déduites en analysant les variables o correspondantes. De la même manière, l'ensemble des liens causaux C est obtenu en examinant chaque variable λ, qui indique comment les préconditions sont satisfaites dans le plan.

3.4 Optimisation du plan

Solveurs CSP souvent intègrent des fonctionnalités d'optimisation, ce qui permet d'optimiser un CSP en spécifiant simplement une fonction objectif.
L'objectif dans Optiplan est de minimiser le *chemin critique* du plan, c'est-à-dire la séquence la plus longue d'actions dépendantes. Pour cela, nous introduisons un ensemble supplémentaire de variables et de contraintes, définies comme suit :

1. Chaque $cell(i,j)$ possède une variable entière de *temps* $s_{i,j}$.
2. Nous définissons une contrainte ternaire sur $(o_{i,j,k}, s_{i,j}, s_{i,k})$, de sorte que si un ordre de tâches $t_j < t_k$ ou $t_j > t_k$ existe, alors une inégalité similaire est imposée entre $s_{i,j}$ et $s_{i,k}$.
3. Nous minimisons la valeur du temps associé à la dernière cellule de L_n (c'est-à-dire l'objectif) : $\min s_{n,|L_n|}$.

Il est important de noter que cette approche garantit l'optimalité dans le problème borné (i.e., avec une profondeur maximale k). Cependant, la véritable solution optimale du problème peut se situer à une profondeur plus grande dans l'arbre de décomposition, au-delà de la borne actuelle. Ainsi, bien que nous trouvions la meilleure solution dans les limites de k, il est possible qu'un plan globalement optimal nécessite une exploration plus profonde.

3.5 Complexité spatiale de l'encodage

Nous évaluons ici la complexité de l'encodage de HiPOD dans le CSP proposé. Supposons que l'arbre contienne N cellules. L'encodage nécessite le nombre suivant de variables :

— **Variables de cellule**. N variables pour représenter les N cellules.
— **Variables d'ordre**. $N * (N-1) \div 2$ variables pour représenter les relations d'ordre entre les cellules, de manière paire.
— **Variables de lien causal**. Le nombre de variables de lien causal est donné par $\sum_{v \in V} \max |\operatorname{pre}(e)|, e \in D(v)$, ce qui prend en compte le nombre maximal de préconditions par cellule.

La plupart des contraintes de notre encodage sont générées par cellule. Considérons une cellule quelconque $cell(i,j)$ dans le HDT :
Contraintes C1 et C5 définissent les relations entre cellules parentes et enfants. Dans le pire des cas, pour chaque $cell(i,j)$, elles génèrent $|D(v_{i,j})| * size(i,j)$ contraintes.
Contraintes C2, C3 et C11 définissent les préconditions d'une cellule et identifient quelles autres cellules les soutiennent. Dans le pire des cas, elles génèrent $size(i,j) * \max |\operatorname{pre}(e)|, e \in D(v_{i,j}) * (N-1)$ contraintes par cellule.
Contrainte C8 gère l'héritage des contraintes d'ordre des cellules parentes vers leurs enfants. Dans le pire des cas, cela peut générer jusqu'à $N - size(i,j)$ contraintes.
Contrainte C9 traite l'ordre des sous-tâches. Dans le pire des cas, où $cell(i,j)$ contient uniquement des méthodes totalement ordonnées, on génère $|D(v_{i,j})| * \sum_{i=1}^{size(i,j)} i$ contraintes par cellule.
Contraintes C4, C6, C7, C12 sont plus simples et ne nécessitent d'être encodées qu'une seule fois.
Maintenant, si l'on considère que

— N est le nombre total de cellules,
— M est la taille de la plus grande décomposition dans tout le problème,
— D est la plus grande taille de domaine rencontrée,
— P est le plus grand nombre de préconditions trouvé pour une action,

alors la complexité globale de l'encodage des contraintes peut être exprimée comme : $(D*M*N + M*P*(N-1)*N + (N-D)*N + D*M + 1)$. Cependant, comme M, D et P augmentent généralement beaucoup plus lentement que N, la complexité peut être généralisée à $\mathcal{O}(N^2)$.

4 Expérimentation

Nous avons développé un planificateur utilisant notre encodage, Optiplan et nous l'avons comparé aux deux planificateurs HTN-POCL : PANDA 3 [6], qui effectue une recherche heuristique basée sur les graphes de décomposition des tâches, [2] et pyHiPOP [15], une adaptation du planificateur HiPOP [2] sans insertion de tâches. Optiplan-N, qui est Optiplan sans fonction d'optimisation, est utilisé pour évaluer l'impact de l'optimisation sur les performances. Les expériences ont été réalisées sur une machine équipée d'un processeur Intel Core i5-1145G7 2.60GHz et d'une mémoire limitée à 10 Go. Chaque instance de problème disposait d'un temps maximal de 10 minutes de calcul CPU.
Nos évaluations couvrent 11 domaines de planification, et les planificateurs ont été évalués selon trois critères principaux : 1) le temps nécessaire pour trouver une solution, 2) la longueur du chemin critique (plus longue séquence d'actions dépendantes dans le plan généré), 3) le nombre de problèmes résolus (couverture).

4.1 Métriques d'évaluation

L'approche générale pour le calcul des scores est inspirée du système de notation de l'International Planning Competition (IPC) pour la catégorie satisficing. Dans ce système, le score de chaque planificateur est évalué relativement au meilleur planificateur et est calculé comme suit : *valeur évaluée*/ *meilleure valeur*, et si un planificateur ne trouve pas de solution, son score est 0.
Le score de benchmark correspond à la somme des scores individuels, un score plus élevé indiquant de meilleures performances relatives. Nous introduisons également un score de benchmark normalisé, calculé comme suit : $\textit{score du planificateur}/max(\textit{scores})$. Contrairement

2. Pour les évaluations, nous avons utilisé la configuration qui a montré les meilleures performances sur les problèmes HTN-POCL (heuristique TDG_c tenant compte des coûts, avec recomputation du TDG activée à chaque décomposition. Nous avons utilisé cette heuristique avec la stratégie de recherche A_2^*).

au scores classiques, dont le maximum théorique correspond à la taille de l'ensemble des problèmes (ce qui peut être irrégulier dans IPC), le score normalisé est toujours compris entre 0 et 1. Cela empêche les planificateurs de tirer avantage d'une bonne performance sur un seul benchmark volumineux.

Notre expérimentation mesure deux scores principaux : 1) *Solve time* (score d'agilité dans IPC), qui est basé sur le temps nécessaire pour trouver une solution. 2) *Qualité du plan*, mesurée en fonction de la longueur du chemin critique du plan généré. De plus, nous indiquons la couverture, c'est-à-dire le nombre de problèmes résolus par chaque planificateur.

4.2 Ensemble de problèmes de benchmark

Parmi les 11 domaines utilisés dans l'évaluation, 8 proviennent de l'ensemble de benchmarks de planification partiellement ordonnée d'IPC 2020 et 2023. Les 3 autres sont de nouveaux domaines de planification que nous avons introduits.

Nous avons défini ces nouveaux domaines car les benchmarks d'IPC sont conçus pour des planificateurs générant des plans séquentiels, laissant peu de possibilités d'exploiter pleinement la planification partiellement ordonnée.

Par exemple, dans *Woodworking*et *UM-Translog*, il est impossible d'obtenir un ordre partiel quelconque. Dans *PCP*, bien que le réseau de tâches initial puisse ne pas contenir de contraintes d'ordre, les plans générés sont toujours séquentiels. Dans *Satellite, Rover, Transport*, les plans partiellement ordonnés sont possibles, mais la concurrence ne vient que de la présence de plusieurs agents, chacun exécutant une séquence stricte d'actions. Nos nouveaux benchmarks visent à mieux tirer parti de la planification partiellement ordonnée. Nous y parvenons en définissant des décompositions avec ordre partiel ou en introduisant plusieurs agents favorisant la concurrence des actions.

Voici un aperçu des nouveaux domaines que nous avons conçus :

— **Oven :** Inspiré d'un scénario industriel réel, ce domaine modélise la cuisson de blocs de ciment dans des fours à températures différentes. Plusieurs blocs peuvent être cuits simultanément dans un four, à condition qu'ils aient la même configuration de température.

— **Medical :** Modélise un scénario collaboratif dans lequel un médecin doit effectuer une chirurgie sur un patient. Avant l'opération, il est nécessaire de préparer les outils et le patient, ce qui nécessite respectivement un et deux infirmiers.

— **Postman :** Similaire au domaine Transport d'IPC 2023, ce domaine implique la livraison de lettres à différentes destinations par un facteur. Tous les lieux sont interconnectés et le facteur n'a aucune limite de capacité pour transporter les lettres.

4.3 Résultats

Un aperçu des résultats est présenté dans Table 1. Nous constatons qu'Optiplan affiche la meilleure couverture avec 9.5/11, démontrant ainsi sa robustesse sur une grande variété de domaines.

Les deux configurations d'Optiplan (avec et sans optimisation) affichent une couverture identique, car lorsque la version optimale atteint la limite de temps, elle retourne toujours la meilleure solution trouvée jusqu'à ce moment, même si elle n'est pas théoriquement optimale.

En revanche, la baisse significative de couverture pour pyHiPOP (3.4) suggère qu'il rencontre des difficultés avec les benchmarks plus flexibles. Cela confirme les observations de [15], selon lesquelles ce planificateur échoue lorsque de nombreux plans ont des valeurs heuristiques très proches – ce qui est précisément le cas des nouveaux benchmarks que nous introduisons.

PANDA 3 affiche une couverture correcte (7.8), mais reste inférieur à Optiplan.

En ce qui concerne le temps de résolution, Optiplan-N est le plus rapide avec un score de 7.7. Cette différence est attendue, car la configuration optimale peut prendre plus de temps à explorer de meilleures solutions.

PANDA 3 est relativement performant avec un score de 6.6, tandis que pyHiPOP est en retrait avec un score de 3.8, confirmant ses difficultés sur les problèmes plus flexibles.

Concernant la qualité des plans (longueur du chemin critique), Optiplan affiche la meilleure performance avec un score de 9.5, démontrant sa capacité à produire des plans de meilleure qualité lorsque le problème le permet. Optiplan-N, bien que toujours compétitif, obtient un score plus faible de 7.9.

PANDA 3 suit avec un score de 6.7, tandis que pyHiPOP reste limité avec un score de 3.4.

En résumé, Optiplan surpasse les autres planificateurs sur tous les critères clés (couverture, temps de résolution et qualité des plans), mettant en avant sa supériorité pour résoudre les problèmes HTN-POCL.

5 Travaux connexes

La combinaison de la planification POCL avec les HTNs permet de générer des plans flexibles, capables de s'adapter aux connaissances expertes de l'utilisateur. Pour cette raison, le rapprochement entre la planification POCL [17, 18] et la planification HTN [12] est un sujet d'intérêt depuis un certain temps.

Le premier système à aborder cette question était DPOCL (Decompositional POCL) [24], mais les deux travaux qui ont réellement formalisé HTN-POCL sont [14, 7]. Tous les travaux suivants se sont développés selon deux approches principales : 1) Enrichir les planificateurs POCL avec HTN, car certaines procédures doivent être respectées strictement. 2) Étendre les planificateurs HTN avec la causalité de POCL, afin d'obtenir des plans plus adaptatifs.

Un exemple de la première approche est le cadre proposé par [14], où les auteurs affirment que les HTN traditionnels réduisent la flexibilité d'un agent face à des situations non

	COUVERTURE			SOLVE TIME				QUALITÉ			
DOMAINE (# de pbs)	*Optiplan* *Optiplan-N*	*pyHiPOP*	*PANDA 3*	*Optiplan*	*Optiplan-N*	*pyHiPOP*	*PANDA 3*	*Optiplan*	*Optiplan-N*	*pyHiPOP*	*PANDA 3*
Satellite (22)	**22/22**	3/22	18/22	15.7	**17.2**	2.5	15.4	**22.0**	20.5	3.0	18.0
Woodworking (30)	5/30	5/30	**10/30**	3.0	2.9	4.7	**6.6**	5.0	5.0	5.0	**10.0**
Smartphone (7)	**5/7**	4/7	**5/7**	1.5	1.8	**4.0**	1.7	**5.0**	**5.0**	4.0	**5.0**
PCP (17)	**11/17**	0/17	8/17	7.6	**10.3**	0.0	5.5	**11.0**	**11.0**	0.0	8.0
UM-Translog (22)	**22/22**	21/22	**22/22**	9.3	9.3	**16.6**	13.9	**22.0**	**22.0**	21.0	**22.0**
Barman-BDI (20)	0/20	**1/20**	0/20	0.0	0.0	**1.0**	0.0	0.0	0.0	**1.0**	0.0
Rover (20)	**4/20**	0/20	**4/20**	1.2	1.2	0.0	**3.6**	**4.0**	0.9	0.0	0.9
Transport (40)	**4/40**	0/40	2/40	**3.6**	**3.6**	0.0	1.6	**4.0**	**4.0**	0.0	2.0
Oven (20)	**8/20**	0/20	4/20	**7.0**	6.6	0.0	3.2	**8.0**	**8.0**	0.0	1.6
Medical (20)	**4/20**	0/20	1/20	**4.0**	**4.0**	0.0	0.2	**4.0**	1.8	0.0	2.0
Postman (20)	**2/20**	0/20	**2/20**	**1.5**	**1.5**	0.0	**1.5**	**2.0**	1.5	0.0	1.5
Total (de 238)	**87.0**	34.0	76.0	54.4	**58.2**	28.8	53.2	**87.0**	79.7	34.0	71.0
Total normalisé (de 11)	**9.5**	3.4	7.8	7.3	**7.7**	3.8	6.6	**9.5**	7.9	3.4	6.7

TABLE 1 – Performances de chaque planificateur sur les 11 benchmarks.

anticipées par l'auteur du domaine. Des travaux similaires ont été menés avec HYBIS [8] et HiPOP [2]. Cependant, ces systèmes permettent une certaine liberté dans la décomposition des tâches abstraites. En revanche, PANDA 3 [6] et pyHiPOP [15] respectent strictement la formalisation HTN, où les tâches abstraites sont uniquement décomposées via des schémas définis par l'utilisateur.

Il convient de noter qu'aucun système n'a tenté d'encoder les problèmes HTN à ordre partiel (PANDA 3 et pyHiPOP s'appuient sur des heuristiques), bien que de nombreuses recherches aient été menées sur ces problématiques dans les contextes POCL et HTN séparément.

Par exemple, parmi les encodages SAT, les contributions notables incluent les travaux de [3, 20, 19]. Ensuite, [22] et [21] ont proposé d'encoder les HTN sous forme de CSP, bien qu'aucun planificateur n'ait été implémenté sur cette base.

Pour être complet, il convient également de mentionner que [11] a proposé un encodage basé sur Answer Set Programming (ASP). Enfin, plusieurs travaux ont cherché à traduire les problèmes HTN en planification classique, comme ceux de [9, 1, 4]. Mais malgré toutes ces avancées, aucun encodage spécifique aux HTN partiellement ordonnés n'avait encore été exploré avant notre travail.

6 Conclusion

Dans ce travail, nous avons introduit une nouvelle technique d'encodage qui fusionne les principes de la planification POCL et HTN afin de résoudre des problèmes de planification hiérarchique partiellement ordonnée.

Notre approche est la première à intégrer une optimisation des plans, constituant ainsi une avancée significative dans le domaine. À travers une évaluation expérimentale sur 11 domaines de planification, nous avons démontré qu'Optiplan offre des performances compétitives en termes de temps de résolution et obtient la meilleure couverture de domaines par rapport aux planificateurs les plus performants.

Nos résultats suggèrent que de futurs travaux pourraient se concentrer sur une réduction supplémentaire de l'espace de recherche, afin d'améliorer l'efficacité du processus de planification. L'exploration de stratégies de recherche alternatives pourrait également conduire à des gains de performance significatifs, étant donné l'impact majeur de la stratégie de recherche sur le comportement global d'un solveur CSP.

De plus, notre encodage introduit la possibilité d'optimiser les plans HTN-POCL en utilisant différentes métriques. Par exemple, il serait possible de générer des plans contenant moins d'actions en maximisant le nombre de cellules vides (assignées à *noop*). Cependant, cette approche pourrait allonger le chemin critique, illustrant ainsi les compromis inhérents à l'optimisation des plans.

En résumé, notre approche représente une avancée majeure dans la planification hiérarchique partiellement ordonnée et ouvre la voie à de nouvelles recherches sur les techniques d'encodage.

Références

[1] Ron Alford, Gregor Behnke, Daniel Höller, Pascal Bercher, Susanne Biundo-Stephan, and David W. 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*, pages 20–28, 2016.

[2] Patrick Bechon, Magali Barbier, Guillaume Infantes, Charles Lesire, and Vincent Vidal. Hipop : Hierarchical partial-order planning. In *Proceedings of the*

European Starting AI Researcher Symposium, pages 51–60, 2014.

[3] Gregor Behnke, Daniel Höller, and Susanne Biundo. totsat - totally-ordered hierarchical planning through sat. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 6110–6118, 2018.

[4] 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*, pages 9687–9697, 2022.

[5] P. Bercher, R. Alford, and D. Höller. A survey on hierarchical planning – one abstract idea, many concrete realizations. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 6267–6275, 2019.

[6] Pascal Bercher, Gregor Behnke, Daniel Höller, and Susanne Biundo. An admissible htn planning heuristic. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 480–488, 2017.

[7] S. Biundo and B. Schattenberg. From abstract crisis to concrete relief – a preliminary report on combining stateabstraction and HTN planning. In *Proceedings of the European Conference on Planning*, page 157–168, 2001.

[8] Luis Castillo, Juan Fdez-Olivares, and Antonio González. On the adequacy of hierarchical planning characteristics for real-world problem solving. In *European conference on planning (ECP-2001)*, pages 169–180, 2001.

[9] N. Cavrel, D. Pellier, and H. Fiorino. Efficient htn to strips encoding for concurrent plans. In *Proceedings of International Conference on Tools with Artificial Intelligence"*, pages 962–969, 2023.

[10] Rina Dechter. *Constraint processing*. Morgan Kaufmann, 2003.

[11] Jürgen Dix, Ugur Kuter, and Dana Nau. Planning in answer set programming using ordered task decomposition. In *Proceedings of the Annual German Conference on AI*, pages 490–504, 2003.

[12] Kutluhan Erol, James A Hendler, and Dana S Nau. Umcp : A sound and complete procedure for hierarchical task-network planning. In *Proceedings of International Conference on AI Planning Systems*, pages 249–254, 1994.

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

[14] Subbarao Kambhampati, Amol Dattatraya Mali, and Biplav Srivastava. Hybrid planning for partially hierarchical domains. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 882–888, 1998.

[15] Charles Lesire and Alexandre Albore. pyhipop - hierarchical partial-order planner. In *Proceedings of the International Planning Competition*, pages 13–16, 2021.

[16] Charles Lesire, Guillaume Infantes, Thibault Gateau, and Magali Barbier. A distributed architecture for supervision of autonomous multi-robot missions. *Autonomous Robots*, 40 :1343–1362, 2016.

[17] David A. McAllester and David Rosenblitt. Systematic nonlinear planning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 634–639, 1991.

[18] J. Scott Penberthy and Daniel S. Weld. UCPOP : A sound, complete, partial order planner for ADL. In *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning*, pages 103–114, 1992.

[19] Dominik Schreiber. Lilotane : A lifted sat-based approach to hierarchical planning. *Journal of Artificial Intelligence Research*, 70 :1117–1181, 2021.

[20] 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*, pages 382–390, 2019.

[21] Tobias Schwartz, Michael Sioutis, and Diedrich Wolter. Towards hierarchical task network planning as constraint satisfaction problem. In *Proceedings of the Workshop on Hierarchical Planning*, pages 78–82, 2022.

[22] Pavel Surynek and Roman Barták. Encoding htn planning as a dynamic csp. In *Proceedings of the International Conference on Principles and Practice of Constraint Programming*, pages 868–868, 2005.

[23] Kevin Tierney, Amanda Jane Coles, Andrew Coles, Christian Kroer, Adam M. Britt, and Rune Møller Jensen. Automated planning for liner shipping fleet repositioning. In *Proceedings of the International Conference on Automated Planning and Scheduling*, pages 279–287, 2012.

[24] R. M. Young and J. D. Moore. Dpocl : A principled approach to discourse planning. *Proceedings of the International Workshop on Natural Langauge Generation*, pages 13–20, 1994.