Codages d'arbres compacts (CTE) pour la planification QBF

Olivier Gasquet, Dominique Longin, Frédéric Maris, Pierre Régnier, Maël Valais

IRIT – University of Toulouse
Toulouse, France

{gasquet,longin,maris,regnier,mael.valais}@irit.fr

Résumé

Les améliorations considérables dans la technologie et les performances des solveurs SAT ont permis de les utiliser pour la résolution de divers problèmes d'intelligence artificielle, et parmi eux, pour le problème de la génération de plans. Récemment, des solveurs QBF (Quantified Boolean Formula) prometteurs ont été développés et nous pouvons nous attendre à ce qu'ils deviennent aussi efficaces que les solveurs SAT dans un proche avenir. Il est donc intéressant d'utiliser le langage QBF qui permet de produire des encodages plus compacts. Nous présentons dans cet article une traduction de problèmes de planification STRIPS en formules propositionnelles quantifiées. Nous proposons deux nouveaux codages d'arbres compacts CTE (Compact Tree Encoding) : CTE-EFA basé sur des frame-axiomes explicatifs dans un espace d'états, et CTE-OPEN basé sur des liens de causalité dans un espace de plans. Nous comparons ensuite ces deux codages à CTE-NOOP qui est proposé dans [3] et qui est basé sur l'utilisation d'actions No-op. En termes de temps d'exécution sur les problèmes de référence, CTE-EFA et CTE-OPEN sont toujours plus efficaces que CTE-NOOP.

Introduction

Une des approches algorithmiques pour la synthèse de plans est la compilation automatique (c'est-à-dire, la transformation) de problèmes de planification. Dans le planificateur SATPLAN [12], un problème de planification est compilé en une formule propositionnelle dont les modèles, correspondant aux plans-solutions, peuvent être trouvés en utilisant un solveur SAT. L'approche SAT recherche un plan solution de longueur fixe k. En cas d'échec cette longueur est augmentée avant de relancer la recherche d'une solution. Dans le cadre classique, décider s'il existe une solution est PSPACE-complet, mais la décision de l'existence d'une solution de taille polynomiale par rapport à la taille du problème est NP-complete [2]. Cette approche par compilation bénéficie directement des améliorations des solveurs SAT [1]. L'exemple le plus marquant est le planificateur BLACKBOX [15, 14] (et ses successeurs SATPLAN'04 [11] et SATPLAN'06 [16]). Ces planificateurs ont obtenu la première place dans la catégorie planificateur optimal (en termes de nombre d'étapes du plan) des compétitions internationales de planification [2] IPC-2004 et IPC-2006. Ce résultat était inattendu car ces planificateurs étaient essentiellement des mises à jour de BLACKBOX et n'incluaient aucune réelle nouveauté : l'amélioration des performances était principalement due aux progrès du solveur SAT sous-jacent.

De nombreuses améliorations de cette approche originale ont été proposées depuis lors, notamment via le développement de codages plus compacts et plus efficaces [13, 7, 19, 18, 25, 26, 27, 28]. Suite à ces travaux, de nombreuses autres techniques similaires pour le codage de problèmes de planification ont été développées : Programmation Linéaire (LP) [31], Problèmes de Satisfaction de Contraintes (CSP) [6], SAT Modulo Theories (SMT) [29, 20, 23]. Plus récemment, une approche QBF (Quantified Boolean Formulas) a été proposée par [22, 3].

Actuellement, les solveurs SAT surpassent les solveurs QBF et l'approche SAT est la plus efficace car les solveurs et les codages ont été grandement améliorés depuis 1992. Cependant, au cours de la dernière décennie, l'approche QBF a suscité un intérêt croissant. L'évaluation compétitive QBFEVAL [3] des solveurs QBF est désormais

1. http://www.satcompetition.org/
2. http://www.icaps-conference.org/index.php/Main/Competitions
3. http://www.qbflib.org/index_eval.php

un événement lié à la conférence internationale SAT et les solveurs QBF s'améliorent régulièrement. QBFEVAL'16 comptait plus de participants que jamais et les articles relatifs à QBF y représentaient 27% de tous les articles publiés à SAT'16. Certaines techniques prometteuses telles que le raffinement d'abstraction guidé par contre-exemple (CEGAR) [4, 10, 9, 21] ont été adaptées à la résolution QBF. Pour des codages SAT / QBF comparables, l'approche QBF présente également l'avantage de générer des formules plus compactes [3]. Actuellement, même si l'approche QBF n'est pas aussi efficace que l'approche SAT, elle mérite l'intérêt de la communauté.

Notre article montre que pour l'approche QBF, au-delà des améliorations sur les solveurs, d'autres travaux doivent être menés pour mettre au point des codages plus performants. Comme nous l'avons indiqué plus haut, cela a été réalisé pour SAT avec des améliorations significatives. Nous présentons ici deux nouveaux codages d'arbres compacts CTE (Compact Tree Encoding) de problèmes de planification en QBF : CTE-EFA basé sur des frame-axiomes explicatifs dans un espace d'états, et CTE-OPEN basé sur des liens de causalité dans un espace de plans. Nous les comparons au codage de l'état de l'art CTE-NOOP basé sur des actions No-op et proposé dans [3]. En termes de temps d'exécution par rapport aux problèmes de référence, CTE-EFA et CTE-OPEN sont toujours plus efficaces que CTE-NOOP.

Planification QBF

Deux approches différentes de la planification QBF ont été proposées par [3] : Le codage plat ou *Flat Encoding* (FE), qui a été introduit par [24] et le codage d'arbre compact ou *Compact Tree Encoding* (CTE) [3] qui surpasse le codage plat. Ces deux codages de planification utilisent la structure de branchement de QBF pour réutiliser un seul ensemble de clauses qui décrit une seule étape dans le plan. Les deux affectations possibles de chaque variable universelle y représentent la première et la seconde moitié du plan réparties autour de cette branche. Les affectations de chaque ensemble existentiel représentent des choix d'action dans une seule étape.

Définitions préliminaires

Soit $\mathcal{F}$ un ensemble fini de *fluents* (propositions atomiques). Un *problème de planification* STRIPS est un triplet $\langle I, \mathcal{A}, G\rangle$ où $I \subseteq \mathcal{F}$ est l'ensemble initial de fluents, $G \subseteq \mathcal{F}$ est l'ensemble des fluents buts et $\mathcal{A}$ est l'ensemble des actions. Une action $a \in \mathcal{A}$ est un triplet $\langle Pre(a), Add(a), Del(a)\rangle$ où

— $Pre(a) \subseteq \mathcal{F}$ est l'ensemble des fluents requis pour exécuter a (les *préconditions* de a),

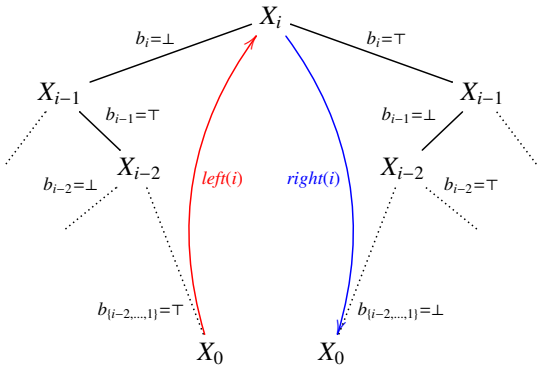

FIGURE 1 – Les deux types de transitions possibles dans un CTE suivant la structure de branchement d'une QBF : $X_0 \rightarrow X_i$ (d'une feuille vers un nœud à gauche) et $X_i \rightarrow X_0$ (d'un nœud vers une feuille à droite). Notons que i fait référence à n'importe quel niveau (excepté pour le niveau feuille), pas seulement la racine.

— $Add(a) \subseteq \mathcal{F}$ et $Del(a) \subseteq \mathcal{F}$ sont les ensembles de fluents respectivement ajoutés et supprimés par l'action a (les *ajouts* et les *retraits* de a).

Tous les codages QBF étudiés dans cet article utilisent des variables propositionnelles pour les actions. Le codage d'arbre compact proposé dans [3] est basé sur le *graphe de planification* introduit dans [1] et utilise des actions No-op supplémentaires comme frame-axiomes. Nous nommerons ce codage CTE-NOOP. Chaque action étant considérée comme une variable propositionnelle, nous définissons un ensemble de variables propositionnelles X, donné par $X = \mathcal{A} \cup \{noop_f \mid f \in \mathcal{F}\}$.

Dans une formule CTE, il faut pouvoir sélectionner deux étapes consécutives du plan afin de définir des transitions (Figure 1). Pour chaque profondeur i de l'arbre, X_i dénote une copie de l'ensemble des variables X.

Pour CTE-NOOP, il existe une seule variable $a_i \in X_i$ pour chaque action et une seule variable $noop_{f,i} \in X_i$ (action No-op) pour chaque fluent utilisée pour déterminer une transition dans le plan. A la même profondeur i, la valeur de ces variables dépend du nœud (correspondant à une étape du plan) sélectionné par les valeurs des variables universelles de branchement supérieures $b_{i+1} \ldots b_k$. Pour plus de détails, on pourra se reporter aux explications données dans les transparents en ligne [4].

Une limite supérieure à la longueur du plan est $2^{k+1} - 1$, où k est le nombre d'alternances de quantificateurs dans la formule booléenne quantifiée associée au problème de planification. Dans le cas d'un CTE, k est aussi la profondeur de l'arbre compact. Le nombre d'états possibles pour un problème de planification donné est limité par $2^{|\mathcal{F}|}$. Ainsi, l'existence d'un plan peut être déterminée en utilisant un

4. https://www.irit.fr/~Frederic.Maris/documents/coplas2018/slides.pdf

codage QBF linéaire avec au plus $k =| \mathcal{F} |$.

Dans la suite, nous proposons deux nouveaux codages de problèmes de planification en QBF. Le premier, noté CTE-OPEN, est basé sur des liens de causalité dans les espaces de plans. Il a été initialement introduit pour SAT par [18] mais doit être adapté en utilisant des variables supplémentaires pour les conditions ouvertes. Le second, noté CTE-EFA, est basé sur des frame-axiomes explicatifs dans les espaces d'états introduits initialement pour SAT par [12] et utilise des variables pour les fluents et les actions.

Codage par liens causaux : CTE-OPEN

Les codages SAT dans les espaces de plans de [18] ne peuvent pas être directement adaptés au CTE. Tous ces codages se réfèrent à trois étapes indexées (pas nécessairement consécutives) du plan, ce qui n'est pas possible dans un CTE car chaque règle ne peut se référer qu'à des noeuds présents sur une même branche de l'arbre. Pour contourner ce problème, il serait possible de dupliquer l'arbre en ajoutant, pour chaque variable de branchement b_i, deux autres variables de branchement b'_i et b''_i, et pour chaque noeud X_i, deux copies de noeud X'_i et X''_i , et les règles d'équivalence $\bigwedge_{x_i \in X_i} ((x_i \leftrightarrow x'_i) \wedge (x_i \leftrightarrow x''_i))$. Malheureusement, cela augmenterait inutilement le facteur de branchement. Nous proposons donc un nouveau codage dans les espaces de plans qui nous permet de ne faire référence qu'à des étapes consécutives du plan.

Pour chaque variable $f \in \mathcal{F}$, nous créons une variable propositionnelle $open_f$ pour exprimer que f se maintient à l'étape précédente et doit être protégé au moins jusqu'à l'étape en cours. Dans la figure 2, la variable f est une *condition ouverte* à l'étape S_i, impliquant que $f \in I$ ou une action a' qui ajoute f est exécutée dans une étape précédente S_{i-k}. Les conditions ouvertes sont propagées vers l'arrière jusqu'à l'état initial ou jusqu'à une étape dans laquelle elles sont ajoutées par une action.

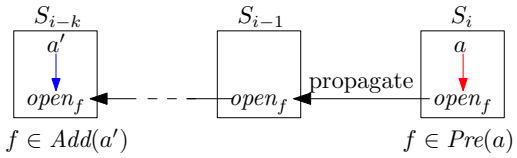

FIGURE 2 – Lien causal : a' produit f pour a.

Nous définissons l'ensemble des variables "open", noté Δ, comme $\Delta = \{open_f \mid f \in \mathcal{F}\}$. Chaque action étant considérée comme une variable propositionnelle, nous définissons un ensemble de variables propositionnelles X, donné par $X = \mathcal{A} \cup \Delta$.

Quantificateurs Pour chaque profondeur i de l'arbre, X_i dénote une copie de l'ensemble des variables X. Il existe une seule variable $a_i \in X_i$ pour chaque action utilisée pour déterminer la dernière transition dans le plan et une seule variable $open_{f,i} \in X_i$ pour chaque fluent utilisée pour déterminer si f est une condition ouverte. A la même profondeur i, la valeur de ces variables dépend du nœud (correspondant à une étape du plan) sélectionné par les valeurs des variables universelles de branchement supérieures $b_{i+1} \dots b_k$.

$$\exists_{a \in \mathcal{A}} a_k . \exists_{f \in \mathcal{F}} open_{f,k} . \forall b_k .$$
$$\exists_{a \in \mathcal{A}} a_{k-1} . \exists_{f \in \mathcal{F}} open_{f,k-1} . \forall b_{k-1} .$$
$$\dots$$
$$\exists_{a \in \mathcal{A}} a_1 . \exists_{f \in \mathcal{F}} open_{f,1} . \forall b_1 . \exists_{a \in \mathcal{A}} a_0 . \exists_{f \in \mathcal{F}} open_{f,0} .$$

Dans ce qui suit, un *nœud* désigne maintenant un nœud qui n'est pas une feuille. Le prédécesseur d'un nœud au niveau i est la feuille la plus à droite du sous-arbre gauche. Le successeur d'un noeud au niveau i est la feuille la plus à gauche du sous-arbre droit. Afin de sélectionner ces transitions, nous introduisons l'opérateur feuille-vers-nœud $left(i)$ défini comme :

$$left(i) \equiv \neg b_i \wedge \bigwedge_{j=1}^{i-1} b_j .$$

Symétriquement, nous introduisons l'opérateur nœud-vers-feuille $right(i)$ défini comme :

$$right(i) \equiv b_i \wedge \bigwedge_{j=1}^{i-1} \neg b_j .$$

Conditions ouvertes Si une action a est exécutée dans une étape du plan, alors chaque précondition de a doit être une condition ouverte à cette étape (c'est-à-dire qu'un lien causal est requis pour cette précondition).

$$\bigwedge_{i=0}^{k} \bigwedge_{a \in \mathcal{A}} \left(a_i \Rightarrow \bigwedge_{f \in Pre(a)} open_{f,i} \right)$$

Dans la dernière étape du plan menant au but (c'est-à-dire la feuille la plus à droite de l'arbre), tous les fluents du but doivent être des conditions ouvertes ou ajoutées par des actions exécutées dans cette étape.

$$\bigwedge_{i=1}^{k} b_i \Rightarrow \bigwedge_{f \in G} \left(open_{f,0} \vee \bigvee_{\substack{a \in \mathcal{A} \\ f \in Add(a)}} a_0 \right)$$

Propagation et fermeture Aucune condition ne doit rester ouverte dans la première étape du plan (c'est-à-dire la

feuille la plus à gauche de l'arbre) si elle n'est pas présente dans l'état initial.

$$\bigwedge_{i=1}^{k} \neg b_i \Rightarrow \bigwedge_{f \in \mathcal{F} \setminus I} \neg open_{f,0}$$

Toute condition ouverte dans une étape doit, soit rester ouverte, soit être ajoutée (fermée) par une action à l'étape précédente.

$$\bigwedge_{i=1}^{k} \bigwedge_{f \in \mathcal{F}} \left(\Big(open_{f,i} \wedge left(i) \Big) \Rightarrow \left(open_{f,0} \vee \bigvee_{\substack{a \in \mathcal{A} \\ f \in Add(a)}} a_0 \right) \right)$$

$$\bigwedge_{i=1}^{k} \bigwedge_{f \in \mathcal{F}} \left(\Big(open_{f,0} \wedge right(i) \Big) \Rightarrow \left(open_{f,i} \vee \bigvee_{\substack{a \in \mathcal{A} \\ f \in Add(a)}} a_i \right) \right)$$

Protection des conditions ouvertes Une condition ouverte dans une étape donnée ne peut pas être supprimée à l'étape précédente. Cela garantit de ne rompre aucun lien de causalité dans le plan.

$$\bigwedge_{i=1}^{k} \bigwedge_{f \in \mathcal{F}} \left(\Big(open_{f,i} \wedge left(i) \Big) \Rightarrow \bigwedge_{\substack{a \in \mathcal{A} \\ f \in Del(a)}} \neg a_0 \right)$$

$$\bigwedge_{i=1}^{k} \bigwedge_{f \in \mathcal{F}} \left(\Big(open_{f,0} \wedge right(i) \Big) \Rightarrow \bigwedge_{\substack{a \in \mathcal{A} \\ f \in Del(a)}} \neg a_i \right)$$

Prévention des interactions négatives Si une action supprime un fluent qui est nécessaire ou est ajouté par une autre action, alors ces deux actions ne peuvent pas être exécutées toutes les deux dans une même étape.

$$\bigwedge_{i=0}^{k} \bigwedge_{a \in \mathcal{A}} \bigwedge_{f \in (Add(a) \cup Pre(a))} \bigwedge_{\substack{a' \in \mathcal{A} \\ a \neq a' \\ f \in Del(a')}} (\neg a_i \vee \neg a'_i)$$

Codage dans les espaces d'états : CTE-EFA

Dans ce codage, nous définissons l'ensemble des variables propositionnelles comme $X = \mathcal{A} \cup \mathcal{F}$. Chaque étape est maintenant définie par une transition (comme dans CTE-OPEN) ainsi que par l'état résultant (valuation des fluents de $\mathcal{F}$). La formule est une adaptation au CTE des règles bien connues de codage SAT de [12] basées sur des frame-axiomes explicatifs dans les espaces d'états.

Quantificateurs A chaque profondeur i de l'arbre, il existe une seule variable a_i pour chaque action utilisée pour déterminer la dernière transition dans le plan et une seule variable f_i pour chaque fluent utilisée pour déterminer l'état. A la même profondeur i, les valeurs de ces variables dépendent du nœud (correspondant à une transition dans le plan et à l'état résultant) sélectionné par les valeurs des variables universelles de branchement supérieures $b_{i+1} \dots b_k$.

$$\begin{array}{c} \exists_{a \in \mathcal{A}} a_k . \exists_{f \in \mathcal{F}} f_k . \forall b_k . \\ \exists_{a \in \mathcal{A}} a_{k-1} . \exists_{f \in \mathcal{F}} f_{k-1} . \forall b_{k-1} . \\ \dots \\ \exists_{a \in \mathcal{A}} a_1 . \exists_{f \in \mathcal{F}} f_1 . \forall b_1 . \exists_{a \in \mathcal{A}} a_0 . \exists_{f \in \mathcal{F}} f_0 . \end{array}$$

But Dans l'état après la dernière transition du plan (c'est-à-dire la feuille la plus à droite de l'arbre), tous les fluents du but doivent être atteints.

$$\bigwedge_{i=1}^{k} b_i \Rightarrow \bigwedge_{f \in G} f_0$$

Conditions et effets des actions Si une action a est exécutée dans une transition du plan, alors chaque effet de a se produit dans l'état résultant et chaque précondition de a est requise dans l'état précédent.

$$\bigwedge_{i=0}^{k} \bigwedge_{a \in \mathcal{A}} \left(a_i \Rightarrow \left(\bigwedge_{f \in Add(a)} f_i \right) \wedge \left(\bigwedge_{f \in Del(a)} \neg f_i \right) \right)$$

$$\bigwedge_{i=1}^{k} \bigwedge_{a \in \mathcal{A}} \left(a_i \wedge left(i) \Rightarrow \bigwedge_{f \in Pre(\mathcal{A})} f_0 \right)$$

$$\bigwedge_{i=1}^{k} \bigwedge_{a \in \mathcal{A}} \left(a_0 \wedge right(i) \Rightarrow \bigwedge_{f \in Pre(\mathcal{A})} f_i \right)$$

De plus, une action qui n'a pas toutes ses préconditions dans l'état initial ne peut pas être exécutée dans la première transition du plan (c'est-à-dire la feuille la plus à gauche de l'arbre) :

$$\bigwedge_{i=1}^{k} \neg b_i \Rightarrow \bigwedge_{\substack{a \in \mathcal{A} \\ Pre(a) \not\subset I}} \neg a_0$$

Frame-axiomes explicatifs Si la valeur d'un fluent change entre deux états consécutifs, alors une action qui produit cette modification est exécutée dans la transition du plan entre ces états.

$$\bigwedge_{i=1}^{k} \bigwedge_{f \in \mathcal{F}} \left((\neg f_0 \wedge f_i \wedge \mathit{left}(i)) \Rightarrow \left(\bigvee_{\substack{a \in \mathcal{A} \\ f \in \mathit{Add}(a)}} a_i \right) \right)$$

$$\bigwedge_{i=1}^{k} \bigwedge_{f \in \mathcal{F}} \left((\neg f_i \wedge f_0 \wedge \mathit{right}(i)) \Rightarrow \left(\bigvee_{\substack{a \in \mathcal{A} \\ f \in \mathit{Add}(a)}} a_0 \right) \right)$$

$$\bigwedge_{i=1}^{k} \bigwedge_{f \in \mathcal{F}} \left((f_0 \wedge \neg f_i \wedge \mathit{left}(i)) \Rightarrow \left(\bigvee_{\substack{a \in \mathcal{A} \\ f \in \mathit{Del}(a)}} a_i \right) \right)$$

$$\bigwedge_{i=1}^{k} \bigwedge_{f \in \mathcal{F}} \left((f_i \wedge \neg f_0 \wedge \mathit{right}(i)) \Rightarrow \left(\bigvee_{\substack{a \in \mathcal{A} \\ f \in \mathit{Del}(a)}} a_0 \right) \right)$$

Une règle supplémentaire est également requise pour décrire les frame-axiomes explicatifs pour la première transition du plan à partir de l'état initial (c'est-à-dire la feuille la plus à gauche de l'arbre) :

$$\bigwedge_{f \in \mathcal{F} \setminus I} \left(\left(f_0 \wedge \bigwedge_{i=1}^{k} \neg b_i \right) \Rightarrow \bigvee_{\substack{a \in \mathcal{A} \\ f \in \mathit{Add}(a) \\ \mathit{Pre}(a) \subset I}} a_0 \right)$$

$$\bigwedge_{f \in I} \left(\left(\neg f_0 \wedge \bigwedge_{i=1}^{k} \neg b_i \right) \Rightarrow \bigvee_{\substack{a \in \mathcal{A} \\ f \in \mathit{Del}(a) \\ \mathit{Pre}(a) \subset I}} a_0 \right)$$

Prévention des interactions négatives Contrairement à CTE-NOOP et CTE-OPEN, les effets contradictoires sont déjà interdits par les règles précédentes (effets des actions). Cette règle doit donc seulement empêcher les interactions entre les préconditions et les retraits des actions. Si une action supprime un fluent nécessaire à une autre action, ces deux actions ne peuvent pas être exécutées dans une même transition du plan.

$$\bigwedge_{i=0}^{k} \bigwedge_{a \in \mathcal{A}} \bigwedge_{f \in \mathit{Pre}(a)} \bigwedge_{\substack{a' \in \mathcal{A} \\ a \neq a' \\ f \in \mathit{Del}(a')}} (\neg a_i \vee \neg a'_i)$$

Tests expérimentaux

Pour comparer ces trois codages sur une même base, nous avons utilisé notre traducteur TouIST [5] [5] qui peut faire appel à différents solveurs. Nous avons utilisé tous les benchmarks IPC STRIPS disponibles (1 à 8, excepté le 7ème que les auteurs ne nous ont pas transmis) sur un CPU Intel® Xeon® E7-8890 v4 @ 2.20GHz, 512 Go de RAM. Les domaines testés incluent Gripper, Logistics, Mystery, Blocks, Elevator, Depots, DriverLog, ZenoTravel, FreeCell, Airport, Pipesworld-NoTankage, Pipesworld-Tankage, PSR, Satellite, OpenStacks, Pathways, Rovers, Storage, TPP, Trucks, ChildSnack, Hiking, VisitAll et le domaine non-IPC Ferry.

Nous avons essayé de considérer autant de solveurs QBF que possible en utilisant QBFEval 2017 comme référence. Qute (version du 09/07/2017, basé sur l'apprentissage par dépendance QCDCL) et CaQE (version du 08/07/2017, basé sur l'abstraction clausale de CEGAR) n'étaient pas capables de donner une valuation pour le quantificateur existentiel externe. AIGSolve et Qell n'étaient pas disponibles au téléchargement. GhostQ n'a pas été utilisé (mais nous aurions dû l'inclure). DepQBF (version 6.03 du 02/08/2017, basé sur la "Q-résolution généralisée", décrite dans [17]) et RAReQS (version 1.1 du 2013-05-07, basé sur une approche CEGAR, détaillée dans [8]) étaient les seuls solveurs restants. RAReQS était toujours deux fois plus rapide que DepQBF, nous avons donc rejeté DepQBF et seulement conservé les résultats pour RAReQS. Enfin, nous n'avons appliqué aucun préprocesseur QBF (par exemple, Bloqqer).

Nous avons testé la résolution de ces benchmarks avec nos nouveaux codages CTE-EFA et CTE-OPEN ainsi qu'avec le codage de l'état de l'art CTE-NOOP. Nous les avons comparés deux par deux en considérant le temps nécessaire pour prouver l'existence d'un plan (temps de décision, Figure 3) et le temps global nécessaire pour obtenir un plan (temps d'extraction, Figure 5). L'étape "décision" consiste à lancer de façon incrémentale le solveur QBF sur un CTE de profondeur croissante jusqu'à ce que le solveur retourne vrai ou atteigne la borne supérieure (nombre total de fluents). L'étape "extraction" consiste en un lancement du solveur par nœud de l'arbre afin de récupérer le plan. Chaque test avait 60 minutes [6] de timeout pour la recherche du plan et 60 minutes pour son extraction. Les résultats complets sont disponibles en ligne dans un fichier Excel [7].

Les résultats montrent que nos codages CTE-EFA et CTE-OPEN sont plus efficaces que CTE-NOOP tant pour décider de l'existence d'un plan que pour l'extraire. CTE-EFA d'un facteur de 2,1 (1/0,4843) et CTE-OPEN d'un facteur de 1,7 (1/0,5953). De plus, la comparaison entre CTE-EFA et CTE-OPEN (Figure 4, Figure 6) montre que CTE-EFA surpasse CTE-OPEN d'un facteur de 1,4 (1/0,7266). La Table 1 donne un résumé des résultats sur les problèmes de référence.

Contrairement aux codages plats, le gain sur CTE-

5. https://www.irit.fr/touist

6. L'étape d'instanciation des actions n'est pas incluse dans le temps écoulé.

7. https://www.irit.fr/~Frederic.Maris/documents/coplas2018/results.xls

Codage	Problèmes résolus	Temps de décision	Littéraux	Clauses	Rapport transitions-sur-nœuds
CTE-NOOP	412 over 2112 (20%)	0%	0%	0%	30%
CTE-EFA	463 over 2112 (22%)	-55%	-26%	+15%	47%
CTE-OPEN	445 over 2112 (21%)	-41%	-2%	-28%	17%

TABLE 1 – Comparaison des codages présentés dans 65 domaines STRIPS des IPC 1 à 8 (sauf IPC 7) avec un total de 2112 problèmes. Le temps de décision, le nombre de littéraux, le nombre de clauses et le rapport transitions-sur-nœuds sont des moyennes. Transitions-sur-nœuds est le rapport moyen du nombre de contraintes basées sur des transitions divisé par le nombre de contraintes basées sur des nœuds (cf. Hypothèse 2, section Discussion ci-après).

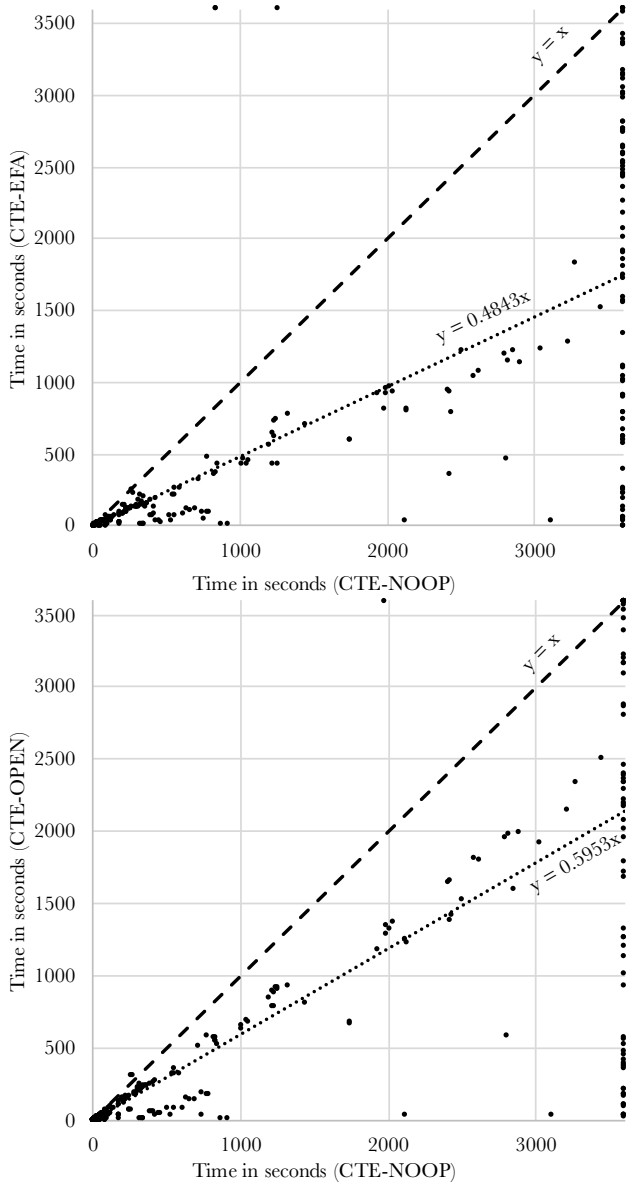

FIGURE 3 – Temps de décision (EFA/OPEN vs NOOP).

NOOP ne peut pas s'expliquer par une différence sur le nombre d'alternance des quantificateurs car la profondeur est la même dans les trois encodages. Cependant, la façon dont les actions sont représentées dans ces encodages pourrait expliquer cette différence.

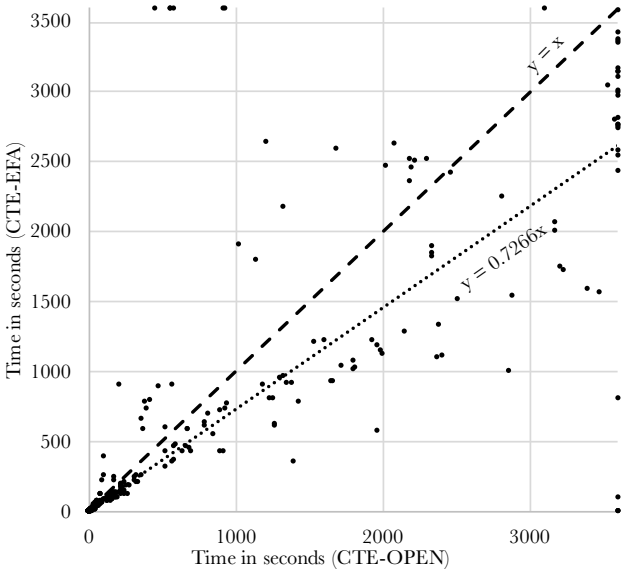

FIGURE 4 – Temps de décision (EFA vs OPEN).

Discussion

Pour tenter d'identifier les causes possibles de ces améliorations, nous avons proposé deux hypothèses, mises ensuite à l'épreuve avec nos tests :

Hypothèse 1 "Le gain de performance est corrélé à une diminution du nombre de clauses et/ou de littéraux à travers les encodages". Bien que la taille du problème soit notoirement non-corrélée à sa difficulté dans SAT, nous nous sommes demandés si nous pourrions voir la même non-corrélation. Comme le montre la Table 1, nous n'observons aucune tendance claire : CTE-EFA tend à avoir un

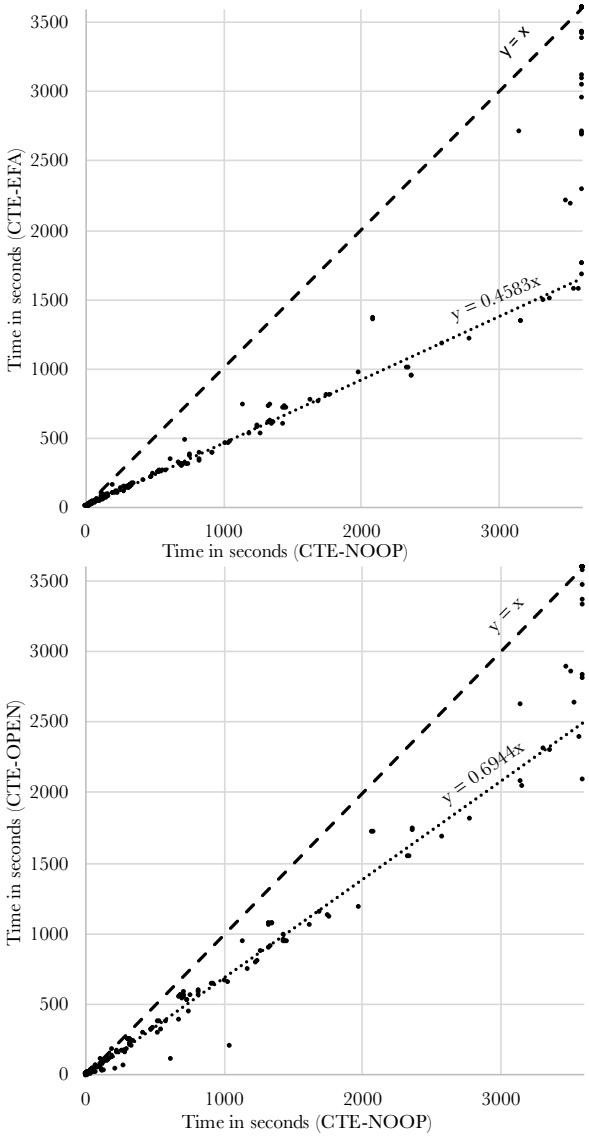

FIGURE 5 – Temps d'extraction (EFA/OPEN vs NOOP).

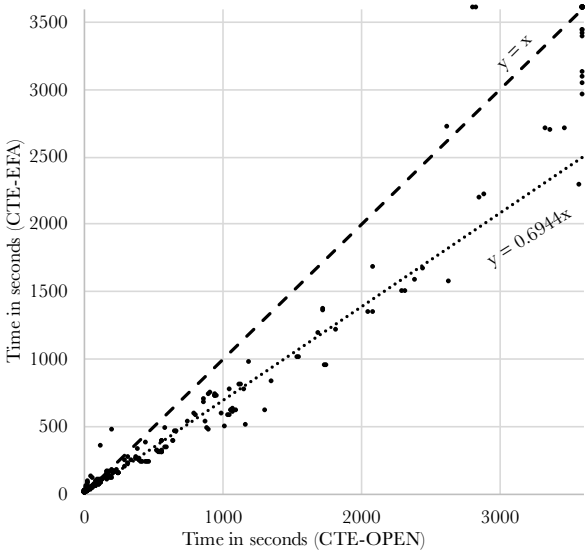

FIGURE 6 – Temps d'extraction (EFA vs OPEN).

nombre de clauses légèrement plus élevé (+15%) que CTE-NOOP bien qu'il ait moins de variables (-26%). CTE-OPEN a le même nombre de littéraux et beaucoup moins de clauses que CTE-NOOP, mais avec un gain de performances inférieur (-41%) à CTE-EFA (-55%). Cette non-corrélation nous a conduit à rejeter cette hypothèse.

Hypothèse 2 "Le gain de performance est dû à une différence dans le nombre de contraintes basées sur une transition par rapport au nombre de contraintes basées sur un nœud". Intuitivement, on peut penser qu'un ratio plus faible de contraintes basées sur les transitions sur les contraintes basées sur les nœuds faciliterait le processus de résolution : dans les contraintes basées sur les nœuds, une clause a le même contexte [8] à travers l'extension QBF entière. Dans les contraintes de branche, la clause correspondante a des contextes différents selon la branche sélectionnée. L'idée est que les clauses basées sur différents contextes ralentissent le solveur. Comme le montre la Table 1, cette hypothèse ne semble pas pouvoir être étayée sur le plan expérimental : bien que CTE-OPEN montre un rapport transitions-sur-nœuds plus faible, il n'aboutit pas au meilleur gain de performance. Au contraire, le rapport CTE-EFA est plus faible, bien qu'il soit le plus efficace par rapport à CTE-NOOP. Nous avons donc aussi rejeté cette explication car nous n'avons noté aucune corrélation la soutenant, malgré une légère tendance où la diminution du temps et le nombre de clauses semblaient corrélées.

Aucune de nos deux hypothèses ne semblant donc être valable, la question de l'explication des gains de performance apportés par nos codages reste ouverte et sera l'objet de nos prochaines recherches.

Conclusion

Nous avons proposé deux nouveaux codages d'arbres compacts pour la planification QBF : CTE-OPEN basé sur des liens causaux (espaces de plans) et CTE-EFA basé sur des frame-axiomes explicatifs (espaces d'états). Nous

8. Contexte et expansion sont définis dans [3]. Intuitivement, l'expansion est un arbre représentant la formule QBF et un contexte est une feuille dans cet arbre.

avons comparé ces codages avec le codage QBF de l'état de l'art CTE-NOOP. En moyenne sur tous les benchmarks IPC STRIPS disponibles, CTE-EFA a été deux fois plus rapide que CTE-NOOP (respectivement 1,7 fois plus rapide pour CTE-OPEN). Grâce à des expérimentations, nous avons rejeté les deux hypothèses que nous avions formulé pour expliquer les causes de cette amélioration : ni la différence de nombre de littéraux et de clauses, ni le rapport transitions-sur-nœuds entre les trois codages ne permettent de tirer de conclusion.

Bien que le travail que nous présentons ici n'apporte pas d'explications sur les raisons de la plus grande efficacité de nos codages, cet article montre l'intérêt qu'il y a à étudier systématiquement les performances de différentes représentations d'actions dans les codages afin de comprendre les choix ontologiques liés à ces représentations.

Un dernier point important montré par notre étude est que le classement des performances respectives des codages SAT est différent de celui des performances des codages QBF. Ainsi, si pour les codages SAT, No-op fonctionne mieux que EFA, dans les codages QBF, CTE-EFA fonctionne mieux que CTE-No-op. Il est donc intéressant de continuer à étudier plus largement les codages SAT pour les comparer à leur homologues QBF afin de déterminer lesquels sont les plus performants et pourquoi.

Références

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

[2] Bylander, T.: *The Computational Complexity of Propositional STRIPS Planning*. Artificial Intelligence, 69(1-2) :165–204, 1994.

[3] Cashmore, M., M. Fox et E. Giunchiglia: *Planning as Quantified Boolean Formula*. Dans *ECAI 2012*.

[4] Clarke, E. M., O. Grumberg, S. Jha, Y. Lu et H. Veith: *Counterexample-guided abstraction refinement for symbolic model checking*. J. ACM, 50(5) :752–794, 2003.

[5] Comte, A., O. Gasquet, A. Heba, O. Lezaud, F. Maris, K. Skander Ben Slimane et M. Valais: *Twist your logic with TouIST*. Dans TTL 2015, CoRR, abs/1507.03663, 2015.

[6] Do, M. B. et S. Kambhampati: *Planning as constraint satisfaction : Solving the planning graph by compiling it into CSP*. Artificial Intelligence, 132(2) :151–182, 2001.

[7] Ernst, M., T. Millstein et D.S. Weld: *Automatic SAT-compilation of planning problems*. Dans *IJCAI 97*, 1997.

[8] Janota, M., W. Klieber, J. Marques-Silva et E. M. Clarke: *Solving QBF with Counterexample Guided Refinement*. Dans *SAT 2012*.

[9] Janota, M., W. Klieber, J. Marques-Silva et E. M. M. Clarke: *Solving QBF with counterexample guided refinement*. Artificial Intelligence, 234 :1–25, 2016.

[10] Janota, M. et J. Marques-Silva: *Solving QBF by Clause Selection*. Dans *IJCAI 2015*.

[11] Kautz, H.: *SATPLAN04 : Planning as Satisfiability*. Dans *International Planning Competition, IPC-04*, 2004.

[12] Kautz, H. et B. Selman: *Planning as satisfiability*. Dans *ECAI 1992*.

[13] Kautz, H. et B. Selman: *Pushing the envelope : Planning, propositional logic and stochastic search*. Dans *AAAI 1996*.

[14] Kautz, H. et B. Selman: *Unifying SAT-based and Graph-based planning*. Dans *IJCAI 1999*.

[15] Kautz, H. et B. Selman: *BLACKBOX : A New Approach to the Application of Theorem Proving to Problem Solving*. Dans *AIPS-98 Workshop on Planning as Combinatorial Search*, 1998.

[16] Kautz, H., B. Selman et J. Hoffmann: *SATPLAN06 : Planning as Satisfiability*. Dans *International Planning Competition, IPC-06*, 2006.

[17] Lonsing, F. et U. Egly: *DepQBF 6.0 : A Search-Based QBF Solver Beyond Traditional QCDCL*. Dans *CADE 26 - International Conference on Automated Deduction, 2017*.

[18] Mali, A. et S. Kambhampati: *On the utility of plan-space (causal) encodings*. Dans *AAAI 1999*.

[19] Mali, A. et S. Kambhampati: *Refinement-based planning as satisfiability*. Dans *AIPS-98 Workshop on Planning as Combinatorial Search*, 1998.

[20] Maris, F. et P. Régnier: *TLP-GP : New Results on Temporally-Expressive Planning Benchmarks*. Dans *ICTAI 2008*.

[21] Rabe, M. N. et L Tentrup: *CAQE : A Certifying QBF Solver*. Dans *FMCAD 2015*.

[22] Rintanen, J.: *Asymptotically Optimal Encodings of Conformant Planning in QBF*. Dans *AAAI 2007*.

[23] Rintanen, J.: *Discretization of Temporal Models with Application to Planning with SMT*. Dans *AAAI 2015*.

[24] Rintanen, J.: *Partial Implicit Unfolding in the Davis-Putnam Procedure for Quantified Boolean Formulae*. Dans *LPAR 2001*.

[25] Rintanen, J.: *Symmetry reduction for SAT representations of transition systems*. Dans *ICAPS 2003*.

[26] Rintanen, J., K. Heljanko et Niemelä: *Parallel encodings of classical planning as satisfiability*. Dans *JELIA 2004*.

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

[28] Rintanen, J., B. Nebel, J. C. Beck et E. A. Hansen (rédacteurs): *Proceedings of the Eighteenth International Conference on Automated Planning and Scheduling, ICAPS 2008, Sydney, Australia, September 14-18, 2008*. AAAI, 2008, ISBN 978-1-57735-386-7.

[29] Shin, J. et E. Davis: *Processes and continuous change in a SAT-based planner*. Artificial Intelligence, 166(1-2) :194–253, 2005.

[30] Tange, O.: *GNU Parallel - The Command-Line Power Tool*. ;login : The USENIX Magazine, 36(1) :42–47, Feb 2011. `http://www.gnu.org/s/parallel`.

[31] Wolfman, S. A. et D. S. Weld: *The LPSAT Engine & Its Application to Resource Planning*. Dans *IJCAI 1999*.