Tables de transposition pour la satisfaction de contraintes

Christophe Lecoutre Lakhdar Sais Sébastien Tabary Vincent Vidal
CRIL−CNRS FRE 2499
Université d'Artois
Lens, France
$\{lecoutre, sais, tabary, vidal\}$@cril.univ-artois.fr

Résumé

Dans ce papier, nous proposons une approche basée sur la reconnaissance d'états dans le cadre de la résolution du problème de satisfaction de contraintes (CSP). L'idée principale consiste en la mémorisation d'états pendant la recherche de manière à prévenir la résolution de sous-réseaux similaires. Les techniques classiques évitent la réapparition de conflits précédemment rencontrés en enregistrant des ensembles conflits (conflict sets). Ceci contraste avec notre approche basée sur les états qui mémorise des sous-réseaux déjà explorés, c'est à dire une photographie de certains domaines sélectionnés. Ces informations sont ensuite exploitées pour éviter soit le parcours d'états inconsistants, soit de recalculer l'ensemble des solutions de ces sous-réseaux. Les deux approches présentent une certaine complémentarité : en effet différents états peuvent être évités à partir d'une même instantiation partielle ou ensemble conflits tandis que différentes instantiations partielles peuvent mener à un même état qui n'a besoin d'être exploré qu'une seule fois. De plus notre méthode permet de détecter et casser dynamiquement certaines formes de symétries (notamment l'interchangeabilité au voisinage). Les résultats expérimentaux obtenus laissent entrevoir des perspectives prometteuses pour la recherche basée sur les états.

Abstract

In this paper, a state-based approach for the Constraint Satisfaction Problem (CSP) is proposed. The key novelty is an original use of state memorization during search to prevent the exploration of similar subnetworks. Classical techniques to avoid the resurgence of previously encountered conflicts involve recording conflict sets. This contrasts with our state-based approach which records subnetworks – a snapshot of some selected domains – already explored. This knowledge is later used to either prune inconsistent states or avoid recomputing the solutions of these subnetworks. Interestingly enough, the two approaches present some complementarity : different states can be pruned from the same partial instantiation or conflict set, whereas different partial instantiations can lead to the same state that needs to be explored only once. Also, our proposed approach is able to dynamically break some kinds of symmetries (e.g. neighborhood interchangeability). The obtained experimental results demonstrate the promising prospects of state-based search.

1 Introduction

Pour les algorithmes de recherche heuristique classique (A*, IDA*, ...) ou les algorithmes de recherche dédiés aux jeux (α-β, SSS* ...), les nœuds de l'arbre de recherche représentent des états du monde et les transitions représentent des mouvements. Aussi, de nombreux états identiques peuvent être rencontrés plusieurs fois à des profondeurs différentes ; ceci étant dû au fait qu'à partir d'un même état initial, différentes séquences de mouvement peuvent mener à des situations du monde identiques. De plus, un état S d'un nœud d'un arbre de recherche rencontré à la profondeur i ne peut mener à une solution meilleure qu'un nœud contenant ce même état S mais rencontré précédemment à une profondeur $j < i$. Certaines parties de l'espace de recherche peuvent donc être explorées plusieurs fois, parfois de manière coûteuse.

Revisiter des états identiques, atteints à partir de différentes séquences de transitions, mieux connus sous le nom de *transpositions*, a été identifié très tôt notamment dans le domaine des jeux d'échec [9, 15, 12]. Une solution à ce problème est d'enregistrer chaque nœud rencontré ainsi que certaines informations pertinentes (e.g. la profondeur, l'évaluation de l'heuristique), dans une *table de transposition*. La structure de données utilisée pour implémenter cette table de transposition est généralement une table de hachage dont la clef est calculée à partir de la description de l'état, par exemple la clef basée sur l'opérateur logique XOR pour les échecs [17]. La quantité de mémoire des machines étant limitée, le nombre d'entrées dans la table de transposition doit être borné. Cette technique a été adaptée aux algorithmes de recherche heuristique tel que IDA* [14] et a été exploitée avec succès dans des planificateurs STRIPS modernes comme par exemple FF [10] et YAHSP [16].

Lorsqu'on s'intéresse à la résolution complète d'un réseau de contraintes, l'utilisation directe des tables de transposition dans un algorithme de recherche avec retours-arrières n'est clairement pas intéressante. Ce type d'algorithme possède une propriété nous assurant qu'un même état d'un réseau de contraintes (c'est à dire les variables, les contraintes et les domaines réduits) ne peut pas être rencontré deux fois au cours de la recherche. En effet, en utilisant par exemple un schéma de branchement binaire, une fois qu'il a été prouvé qu'une décision positive $X = v$ mène à une contradiction, la décision opposée $X \neq v$ est immédiatement prise sur l'autre branche. En d'autres termes, dans la première branche le domaine de la variable X est réduit à la valeur singleton $\{v\}$, tandis que dans la seconde branche v est supprimé du domaine de X : évidemment, aucun état ou la décision $X = v$ a été prise ne peut être identique à un état ou $X \neq v$ est vrai.

Nous montrons cependant dans ce papier que deux réseaux P_1 et P_2, obtenus à partir de deux séquences différentes de décisions effectuées pendant la résolution d'un réseau de contraintes P peuvent être réduits au même *sous-réseau*. Nous proposons trois *opérateurs de réduction* satisfaisant la propriété suivante : P_1 est satisfiable si et seulement si P_2 est satisfiable. Ces opérateurs suppriment certaines variables sélectionnées ainsi que les contraintes les impliquant. Par exemple, suivant l'opérateur utilisé, les variables apparaîssant dans la portée de contraintes universelles ou ayant un domaine associé qui n'a pas encore été filtré peuvent être supprimées. En pratique, lorsqu'un sous-réseau a été identifié, on peut l'enregistrer dans une table de transposition, qu'il suffira de contrôler avant l'ouverture de chaque nœud. Une recherche dans cette table permettra alors d'éviter l'exploration d'un réseau, réduit à un sous-réseau déjà exploré et présent dans la table.

2 Préliminaires

Un réseau de contraintes (CN) P est un couple $(\mathscr{X}, \mathscr{C})$ où $\mathscr{X}$ est un ensemble fini de n variables et $\mathscr{C}$ un ensemble fini de e contraintes. Chaque variable $X \in \mathscr{X}$ possède un domaine associé, noté $dom^P(X)$ ou simplement $dom(X)$, qui contient l'ensemble des valeurs autorisées pour X. L'ensemble des variables de P sera noté $vars(P)$. Une instantiation t d'un ensemble $\{X_1, ..., X_q\}$ de variables est un ensemble $\{(X_i, v_i) \mid i \in [1, q] \wedge v_i \in dom(X_i)\}$. La valeur v_i assignée à X_i dans t sera notée par $t[X_i]$. Chaque contrainte $C \in \mathscr{C}$ implique un sous ensemble de variables de $\mathscr{X}$, appelé portée et noté $scp(C)$, et possède une relation associée, notée $rel(C)$, qui représente l'ensemble des instantiations autorisées pour les variables de sa portée. Une solution de P est une instantiation de $vars(P)$ telle que toutes les contraintes sont satisfaites. L'ensemble de toutes les solutions de P est noté $sol(P)$, et P est satisfiable ssi $sol(P) \neq \emptyset$.

Le problème de satisfaction de contraintes (CSP pour Constraint Satisfaction Problem) qui consiste à déterminer si un réseau de contraintes donné est satisfiable, est NP-complet. Une instance CSP est alors définie par un réseau de contraintes, et la résoudre consiste à trouver une (ou plusieurs) solution(s) ou alors à prouver son insatisfiabilité. Pour résoudre une instance CSP, on peut modifier le CN en utilisant des méthodes d'inférence ou de recherche. Très souvent, les domaines des variables sont réduits en supprimant des valeurs inconsistantes, c'est à dire des valeurs qui ne peuvent apparaître dans aucune solution. En effet il est possible de filtrer les domaines en tenant compte de certaines propriétés du réseau de contraintes. La consistance d'arc généralisée (GAC) reste la propriété principale. GAC est maintenu pendant la recherche par l'algorithme MGAC, appelé MAC dans le cas binaire.

A partir de maintenant, on considérera un opérateur d'inférence ϕ assurant la consistance en filtrant les domaines [4] et pouvant être employé à chaque étape d'un arbre de recherche. Étant donné un réseau de contraintes P et un ensemble de décisions Δ, $P|_\Delta$ est le réseau de contraintes dérivé de P tel que, pour chaque décision positive $(X = v) \in \Delta$, $dom(X)$ est restreint à $\{v\}$, et pour chaque décision négative $(X \neq v) \in \Delta$, v est supprimé de $dom(X)$. $\phi(P)$ est le réseau de contraintes dérivé de P, obtenu après application d'un opérateur d'inférence ϕ. Si une variable avec un domaine vide apparaît dans $\phi(P)$ alors P est clairement insatisfiable, noté $\phi(P) = \bot$.

Une contrainte est *universelle* si toutes les instantiations construites à partir du domaine de ses variables satisfont cette contrainte.

Définition 1 *Soit $P = (\mathscr{X}, \mathscr{C})$ un CN. Une contrainte $C \in \mathscr{C}$ avec $scp(C) = \{X_1, \ldots, X_r\}$ est universelle ssi $\forall v_1 \in dom(X_1), \ldots, \forall v_r \in dom(X_r), \exists t \in rel(C)$ tel que $t[X_1] = v_1, \ldots, t[X_r] = v_r$.*

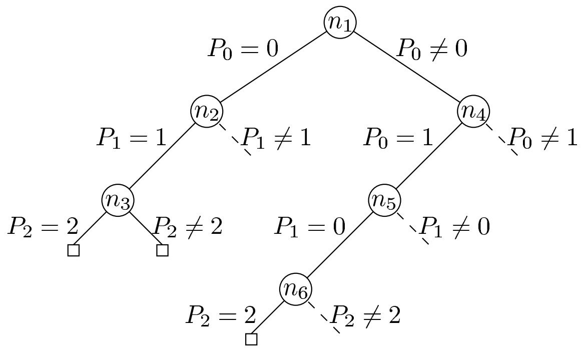
Un *sous-réseau de contraintes* peut être obtenu à partir d'un CN en supprimant de celui-ci un sous-ensemble de ses variables ainsi que les contraintes les impliquant.

Définition 2 *Soit* $P = (\mathscr{X}, \mathscr{C})$ *un* CN *et* $S \subseteq \mathscr{X}$. *Le sous-réseau de contraintes* $P \ominus S$ *est le CN* $(\mathscr{X}', \mathscr{C}')$ *tel que* $\mathscr{X}' = \mathscr{X} \setminus S$ *et* $\mathscr{C}' = \{C \in \mathscr{C} \mid scp(C) \cap S = \emptyset\}$.

3 Illustration

Le problème académique des *pigeons* présenté ici avec cinq pigeons illustre notre propos. Ce problème est composé de cinq variables $P_0, \ldots, P_4$ représentant les pigeons, et dont le domaine initialement égal à $\{0, \ldots, 3\}$ représente les numéros des boites dans lesquelles on place les pigeons. Les contraintes de ce problème interdisent de mettre deux pigeons dans la même boîte, le rendant ainsi insatisfiable puisqu'il y a cinq pigeons et seulement quatre boîtes. Ces contraintes peuvent être exprimées avec une clique de contraintes binaires : $P_0 \neq P_1, P_0 \neq P_2, \ldots, P_1 \neq P_2, \ldots$ La figure 1 décrit une vue partielle d'un arbre de recherche pour ce problème construit par MAC avec un schéma de branchement binaire ainsi que l'état des domaines à chaque nœud de cet arbre.

Remarquons tout d'abord que les six nœuds $n_1, \ldots, n_6$ représentent des réseaux qui sont tous différents, étant donné que les domaines de ces variables diffèrent d'au moins une valeur. Cependant si on s'attarde sur les nœuds n_3 et n_6, on s'aperçoit que les seules différences portent sur les variables P_0 et P_1 dont les domaines sont respectivement réduits aux singletons $\{0\}$ et $\{1\}$ dans n_3, et $\{1\}$ et $\{0\}$ dans n_6. Le domaine des autres variables P_2, P_3 et P_4 est égal à $\{2, 3\}$. Les réseaux associés aux nœuds n_3 et n_6 sont représentés sur la figure 2, incluant l'ensemble des instantiations autorisées pour chaque contrainte. La structure de ces deux réseaux est très similaire, la seule différence étant l'inversion des valeurs 0 et 1 entre P_0 et P_1.



$$n_1: \begin{bmatrix} P_0: \{0,1,2,3\} \\ P_1: \{0,1,2,3\} \\ P_2: \{0,1,2,3\} \\ P_3: \{0,1,2,3\} \\ P_4: \{0,1,2,3\} \end{bmatrix} \quad n_2: \begin{bmatrix} P_0: \{0\} \\ P_1: \{1,2,3\} \\ P_2: \{1,2,3\} \\ P_3: \{1,2,3\} \\ P_4: \{1,2,3\} \end{bmatrix} \quad n_3: \begin{bmatrix} P_0: \{0\} \\ P_1: \{1\} \\ \mathbf{P_2: \{2,3\}} \\ \mathbf{P_3: \{2,3\}} \\ \mathbf{P_4: \{2,3\}} \end{bmatrix}$$

$$n_4: \begin{bmatrix} P_0: \{1,2,3\} \\ P_1: \{0,1,2,3\} \\ P_2: \{0,1,2,3\} \\ P_3: \{0,1,2,3\} \\ P_4: \{0,1,2,3\} \end{bmatrix} \quad n_5: \begin{bmatrix} P_0: \{1\} \\ P_1: \{0,2,3\} \\ P_2: \{0,2,3\} \\ P_3: \{0,2,3\} \\ P_4: \{0,2,3\} \end{bmatrix} \quad n_6: \begin{bmatrix} P_0: \{1\} \\ P_1: \{0\} \\ \mathbf{P_2: \{2,3\}} \\ \mathbf{P_3: \{2,3\}} \\ \mathbf{P_4: \{2,3\}} \end{bmatrix}$$

FIG. 1 – Problème des Pigeons : vue partielle de l'arbre de recherche

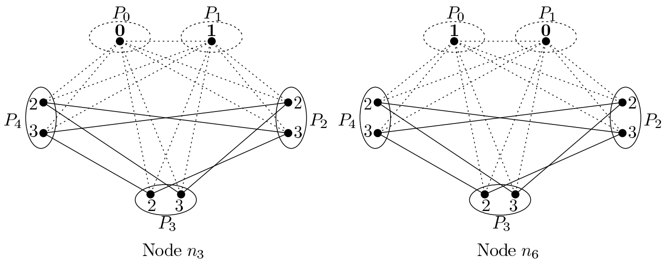


FIG. 2 – Problème des Pigeons : 2 sous-réseaux similaires

Les deux points cruciaux concernant les nœuds n_3 et n_6 sont les suivants : (1) P_0 et P_1 ne joueront plus aucun rôle dans la recherche associée à ces deux nœuds, et (2) vérifier la satisfiabilité du réseau attaché au nœud n_3 est équivalent à vérifier la satisfiabilité de n_6. Le point (1) est facile à constater : comme la consistance d'arc (AC) est maintenue, toutes les contraintes impliquant P_0 et P_1 sont universelles ; quelque soit la valeur assignée aux autres variables, ces contraintes seront satisfaites. Les variables P_0 et P_1 peuvent alors être déconnectées du réseau de contraintes associé aux nœuds n_3 et n_6. Le point (2) est alors également vrai : les deux sous-réseaux de contraintes impliquant les variables restantes P_2, P_3 et P_4 ainsi que les contraintes les impliquant sont égaux, et par conséquent n_3 est satisfiable si et seulement si n_6 est satisfiable. Si après avoir prouvé l'insatisfiablilité de n_3, le sous-réseau extrait à partir de celui-ci avait été enregistré dans une table de transposition, nous aurions pu éviter l'exploration de n_6 en contrôlant la table avant d'explorer le nœud.

4 Identifier des sous-réseaux

Enregistrer des états complets n'a pas d'intérêt puisque ceux-ci ne peuvent pas être rencontrés plusieurs fois durant la recherche effectuée par un algorithme tel que MGAC. On peut cependant identifier des sous-réseaux qui peuvent être atteints plusieurs fois dans un même arbre de recherche. De tels sous-réseaux devraient être idéalement aussi généraux que possible, et ne consommer qu'une petite quantité de mémoire. Nous présentons dans cette section trois opérateurs de réduction dont l'objectif est essentiellement de minimiser le nombre de variables enregistrées. Intuitivement étant donné un sous-réseau, plus le nombre de variables éliminées est important, meilleure est la capacité d'élagage de ce sous-réseau. Cela contribue également à réduire la quantité de mémoire nécessaire pour stocker ces sous-réseaux.

4.1 Préserver les solutions (opérateur ρ^{sol})

Le premier opérateur, noté ρ^{sol}, préserve l'ensemble des solutions d'un réseau de contraintes donné. Appliqué sur un réseau, il permet de supprimer les variables appelées *s-éliminables* qui possèdent un domaine singleton et n'apparaîssent que dans des contraintes universelles.

Définition 3 *Soit $P = (\mathscr{X}, \mathscr{C})$ un CN. Une variable $X \in \mathscr{X}$ est s-éliminable ssi $|dom(X)| = 1$ et $\forall C \in \mathscr{C} \mid X \in scp(C)$, C est universelle. L'ensemble des variables s-éliminables de P est noté $S_{elim}(P)$.*

L'opérateur ρ^{sol} supprime toutes les variables s-éliminables d'un CN ainsi que les contraintes impliquant au moins l'une d'entre elles.

Définition 4 *Soit P un CN. $\rho^{sol}(P) = P \ominus S_{elim}(P)$.*

Les solutions d'un réseau P peuvent être énumérées en étendant les solutions du sous-réseau $\rho^{sol}(P)$ avec l'interprétation construite à partir des variables s-éliminables. En effet, le domaine des variables éliminées est singleton, et les contraintes supprimées n'ont plus d'impact sur le réseau.

Proposition 1 *Soit P un CN et $t = \{(X, v) \mid X \in S_{elim}(P) \wedge dom(X) = \{v\}\}$. $sol(P) = \{s \cup t \mid s \in sol(\rho^{sol}(P))\}$.*

Preuve. N'importe quelle solution de P satisfait également les contraintes de $\rho^{sol}(P)$. La réciproque est immédiate : n'importe quelle solution s de $\rho^{sol}(P)$ peut être étendue à une unique solution de P puisque les variables éliminées (de $S_{elim}(P)$) ont un domaine singleton et les contraintes éliminées sont universelles. □

4.2 Préserver la satisfiabilité (opérateur ρ^{uni})

Le second opérateur, noté ρ^{uni}, préserve la satisfiabilité d'un réseau de contraintes (mais pas nécessairement toutes ses solutions). Appliqué à un réseau, il supprime les variables *u-éliminables*. Ces variables apparaîssent seulement dans des contraintes universelles.

Définition 5 *Soit $P = (\mathscr{X}, \mathscr{C})$ un CN. Une variable $X \in \mathscr{X}$ est u-éliminable ssi $\forall C \in \mathscr{C} \mid X \in scp(C)$, C est universelle. L'ensemble des variables u-éliminables de P est noté par $U_{elim}(P)$.*

L'opérateur ρ^{uni} supprime les variables u-éliminables d'un réseau de contraintes ainsi que les contraintes impliquant au moins l'une de ces variables.

Définition 6 *Soit P un CN. $\rho^{uni}(P) = P \ominus U_{elim}(P)$.*

L'opérateur ρ^{uni} préserve la satisfiabilité puisque les contraintes éliminées n'ont plus d'impact sur le réseau (comme elles sont universelles). Quelque soit le réseau P, comme les variables s-éliminables sont également u-éliminables, $\rho^{uni}(P)$ est un sous-réseau de $\rho^{sol}(P)$. ρ^{sol} peut donc être considéré comme un cas particulier de ρ^{uni}.

Proposition 2 *Un réseau de contraintes P est satisfiable si et seulement si $\rho^{uni}(P)$ est satisfiable.*

Preuve. Supprimer n'importe quelle contrainte universelle ne change pas la satisfiabilité d'un réseau. L'opérateur ρ^{uni} ne supprime que des contraintes universelles et les variables du réseau qui deviennent déconnectées. □

4.3 Réduire les sous-réseaux (opérateur ρ^{red})

Le troisième opérateur, noté ρ^{red}, extrait un sous-réseau réduit à partir d'un réseau courant en éliminant les variables u-éliminables ainsi que les variables appelées *r-éliminables*. Ces dernières correspondent aux variables dont le domaine est resté inchangé après avoir pris un ensemble de décisions et appliqué un opérateur d'inférence.

Définition 7 *Soit $P = (\mathscr{X}, \mathscr{C})$ un CN, ϕ un opérateur d'inférence, Δ un ensemble de décisions et $P' = \phi(P|_\Delta)$. Une variable $X \in \mathscr{X}$ est r-éliminable dans P' vis-à-vis de P ssi $dom^{P'}(X) = dom^P(X)$. L'ensemble des variables r-éliminables de P' est noté $R^P_{elim}(P')$.*

Définition 8 *Soit $P = (\mathscr{X}, \mathscr{C})$ un CN, ϕ un opérateur d'inférence, Δ un ensemble de décisions et $P' = \phi(P|_\Delta)$. $\rho^{red}(P') = \rho^{uni}(P') \ominus R^P_{elim}(P')$.*

Le résultat principal de ce papier montre que deux réseaux dérivés d'un même réseau initial peuvent être réduits au même sous-réseau, la satisfiabilité de l'un déterminant la satisfiabilité de l'autre.

Proposition 3 *Soit P un CN, ϕ un opérateur d'inférence, et Δ_1, Δ_2 deux ensembles de décisions. Soit $P_1 = \phi(P|_{\Delta_1})$ et $P_2 = \phi(P|_{\Delta_2})$. Si $\rho^{red}(P_1) = \rho^{red}(P_2)$, alors P_1 est satisfiable si et seulement si P_2 est satisfiable.*

Preuve. Montrons que $sol(P_1) \neq \emptyset \Rightarrow sol(P_2) \neq \emptyset$. A partir de n'importe quelle solution $s \in sol(P_1)$, on peut construire une solution $s' \in sol(P_2)$ comme suit : si $s[X] \in dom^{P_2}(X)$ alors $s'[X] = s[X]$, sinon $s'[X] = a$ où a est n'importe quelle valeur de $dom^{P_2}(X)$. Pour n'importe quelle contrainte C de P, nous savons que s satisfait C. Montrons maintenant que s' satisfait également C. Il est évident que ceci est vrai si $\nexists X \in scp(C) \mid s[X] \neq s'[X]$. Si maintenant $\exists X \in scp(C) \mid s[X] \neq s'[X]$, nous pouvons montrer que C est universelle, et par conséquent, C est satisfaite par s'. En effet, nous avons montré auparavant que $s'[X] \neq s[X]$ implique $X \in U_{elim}(P_2)$ à partir duquel nous en déduisons que n'importe quelle contrainte impliquant X est universelle (par définition de U_{elim}).

Supposons que $X \in vars(\rho^{uni}(P_2))$ (et alors, $X \notin U_{elim}(P_2)$). Si $X \in R^P_{elim}(P_2)$ alors il est immédiat que $s[X] \in dom^{P_2}(X)$ (et donc, $s'[X] \neq s[X]$ est impossible). Sinon, cela veut dire que X appartient à $\rho^{red}(P_2)$, et par conséquent appartient également à $\rho^{red}(P_1)$. Comme le domaine des variables de ces deux sous-réseaux sont égaux (par hypothèse), par construction de s', on ne peut pas avoir $s'[X] \neq s[X]$. On peut donc en conclure que $X \in U_{elim}(P_2)$. Finalement, en appliquant le même raisonnement on peut montrer que $sol(P_2) \neq \emptyset \Rightarrow sol(P_1) \neq \emptyset$. □

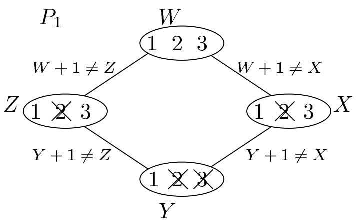
Proposition 4 *Soit P un réseau de contraintes binaires, Δ un ensemble de décisions, et $P' = (\mathscr{X}', \mathscr{C}') = \rho^{red}(\mathrm{AC}(P|_\Delta))$. Nous avons alors : $\forall X \in \mathscr{X}', 1 < |dom^{P'}(X)| < |dom^P(X)|$.*

Preuve. Une variable X avec un domaine singleton est supprimée si toutes les contraintes impliquant X sont universelles. En effet, comme on applique la consistance d'arc (AC), toutes les valeurs pour n'importe quelle variable Y connectée à X (les contraintes étant binaires) sont compatibles avec la seule valeur de X. Par définition de ρ^{red}, une variable X telle que $|dom^{P'}(X)| = |dom^P(X)|$ est éliminée. □

5 Capacité d'élagage des opérateurs de réduction

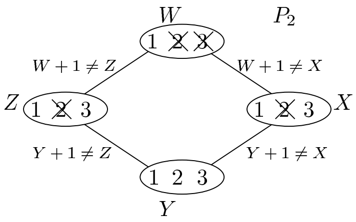
Dans le contexte de la détermination de la satisfiabilité d'une instance CSP, nous discutons maintenant de la capacité d'élagage des opérateurs que nous avons définis. Clairement, plus le nombre de variables éliminées à partir d'un réseau P_1 pour produire un sous-réseau P'_1 est important, plus P'_1 élaguera de réseaux. En effet, si un réseau P_2 peut être évité parce que celui-ci peut être réduit à P'_1, alors les domaines des variables de P_2 n'apparaîssant pas dans P'_1 peuvent être dans n'importe quel état : ils peuvent soit contenir toutes les valeurs vis-à-vis du problème initial, ou alors être réduits de manière à n'apparaître que dans des contraintes universelles. Il est facile de voir que par définition des opérateurs de réduction, $vars(\rho^{red}(P)) \subseteq vars(\rho^{uni}(P)) \subseteq vars(\rho^{sol}(P))$. On peut donc espérer avoir un meilleur (au moins égal) pouvoir d'élagage avec l'opérateur ρ^{red} qu'avec les opérateurs ρ^{sol} et ρ^{uni}.

Les réseaux présentés sur la figure 3 illustrent ce phénomène. Sur le premier réseau, l'assignation $Y = 1$ à partir d'un réseau initial P (où les domaines sont tous égaux à $\{1, 2, 3\}$) laisse le domaine de W inchangé et supprime la valeur 2 de $dom(X)$ et $dom(Z)$. Ceci mène alors au réseau dérivé $P_1 = \mathrm{AC}(P|_{Y=1})$. Sur le second réseau, l'assignation $W = 1$ sur P, laisse le domaine de Y inchangé et supprime la valeur 2 de $dom(X)$ et $dom(Z)$, menant au réseau dérivé $P_2 = \mathrm{AC}(P|_{W=1})$. Tandis que ρ^{uni} produit deux sous-réseaux différents à partir de P_1 et P_2, l'opérateur ρ^{red} fournit le même sous-réseau réduit, quelque soit la raison de la suppression d'une variable (u-éliminable ou r-éliminable). Nous savons donc grâce à la proposition 3 qu'une fois l'exploration de P_1 ou P_2 effectuée, l'exploration du second ne sera pas utile pour déterminer sa satisfiabilité.



$$\rho^{sat}(P_1) = \begin{bmatrix} W : \{1,2,3\} \\ X : \{1,3\} \\ Z : \{1,3\} \end{bmatrix}$$

$$\rho^{red}(P_1) = \begin{bmatrix} X : \{1,3\} \\ Z : \{1,3\} \end{bmatrix}$$



$$\rho^{sat}(P_2) = \begin{bmatrix} X : \{1,3\} \\ Y : \{1,2,3\} \\ Z : \{1,3\} \end{bmatrix}$$

$$\rho^{red}(P_2) = \begin{bmatrix} X : \{1,3\} \\ Z : \{1,3\} \end{bmatrix}$$

FIG. 3 – Réduction de réseaux par ρ^{uni} et ρ^{red}

6 Algorithme de recherche exploitant l'opérateur ρ^{red}

Dans cette section, nous présentons succinctement un algorithme effectuant une recherche en profondeur d'abord et maintenant une consistance ϕ en filtrant les domaines (au moins, assurer que les contraintes impliquant des variables avec un domaine singleton sont satisfaites). Cet algorithme élague les états inconsistants grâce à l'opérateur ρ^{red}. L'idée principale est d'enregistrer dans une table de transposition tous les sous-réseaux réduits extraits à partir de nœuds prouvés inconsistants. Les nœuds dont le sous-réseau réduit se trouve déjà dans la table de transposition peuvent alors être évités.

La fonction récursive $solve$ détermine la satisfiabilité d'un réseau P (voir algorithme 1). A une étape donnée, si le réseau courant (après application de ϕ) est inconsistant, la valeur $false$ est retournée (ligne 2) tandis que si toutes les variables ont été assignées, la valeur $true$ est retournée (ligne 3). Dans le cas contraire, nous vérifions si le nœud courant ne peut pas être élagué grâce à la table de transposition (ligne 4). Si la recherche se poursuit, on sélectionne un couple (X, a) et on appelle récursivement la fonction $solve$ en considérant deux branches : l'une étiquetée par $X = a$ et la seconde par $X \neq a$ (lignes 5 et 6). Si une solution est trouvée, la valeur $true$ est retournée, sinon le réseau courant a été prouvé inconsistant et le sous-réseau réduit qui lui correspond est ajouté dans la table de transposition (ligne 7) avant de retourner la valeur $false$.

Cet algorithme peut être légèrement modifié pour énumérer toutes les solutions d'un réseau en utilisant l'opérateur ρ^{sol}. Pour cela, la table de transposition doit contenir tous les sous-réseaux rencontrés (et non seulement ceux prouvés insatisfiables) ainsi qu'une information supplémentaire : l'ensemble solution. Quand un réseau P est tel que $\rho^{sol}(P)$ est déjà présent dans la table, les solutions de P peuvent être étendues à partir des solutions de $\rho^{sol}(P)$ enregistrées dans la table, grâce à l'interprétation construite à partir des variables s-éliminables de P (c.f. proposition 1). De manière similaire, on peut compter le nombre de solutions d'un problème en associant à chaque sous-réseau réduit enregistré dans la table, le nombre de ses solutions.

Algorithm 1 $solve(P_{init} = (\mathscr{X}, \mathscr{C})$: CN) : Booléen

1: $P = \phi(P_{init})$
2: **if** $P = \bot$ **then** return $false$
3: **if** $\forall X \in \mathscr{X}, |dom(X)| = 1$ **then** return $true$
4: **if** $\rho^{red}(P) \in transposition\_table$ **then** return $false$
5: choisir un couple (X, a) avec $|dom(X)| > 1 \wedge a \in dom(X)$
6: **if** $solve(P|_{X=a})$ ou $solve(P|_{X \neq a})$ **then** return $true$
7: ajouter $\rho^{red}(P)$ à $transposition\_table$
8: return $false$

7 Portée et connexions

Notre approche est à mettre en relation avec plusieurs domaines de recherche de la programmation par contraintes. En effet la recherche basée sur les états permet d'éliminer automatiquement certaines formes de symétries pendant la recherche, et présente une forte complémentarité avec l'enregistrement de nogoods.

L'interchangeabilité au voisinage est une forme affaiblie de l'interchangeabilité [8] qui peut être exploitée en pratique pour réduire l'espace de recherche. Étant donnée une variable X, deux valeurs a et b de $dom(X)$ sont interchangeables au voisinage ssi pour n'importe quelle contrainte C impliquant X, l'ensemble des supports de a pour X dans C est égal à l'ensemble des supports de b pour X dans C. Nous pouvons observer que notre approche basée sur les états évite les états redondants provenant de valeurs interchangeables. En effet, si P est un réseau tel que les valeurs a et b pour une variable X de P sont interchangeables, il apparaît clairement que les sous-réseaux extraits de $P|_{X=a}$ et $P|_{X=b}$ sont identiques après application de n'importe quel opérateur de réduction ρ.

L'interchangeabilité est à mettre en relation avec la notion de symétrie [3] dont l'objectif est d'éviter de parcourir des parties de l'arbre de recherche qui sont symétriques à des sous-arbres précédemment explorés. Ceci peut mener à une réduction importante de l'espace de recherche requis pour résoudre un réseau de contraintes. Pour atteindre un tel objectif, on doit tout d'abord identifier les symétries puis les exploiter. Différentes approches ont été proposées pour les exploiter ; la plus proche de notre contexte étant "symmetry breaking via dominance detection" (SBDD).

Le principe de SBDD est le suivant : à chaque nouveau nœud atteint par l'algorithme de recherche, on vérifie si ce nœud est équivalent ou dominé par un nœud qui a déjà été exploré auparavant. Cette approche requiert (1) la mémorisation d'information au sujet des nœuds explorés pendant la recherche et (2) l'exploitation de cette information en utilisant tout ou partie des symétries obtenues à partir du groupe de symétries associé au réseau initial. L'information enregistrée pour un nœud peut être le domaine courant de toutes les variables du réseau initial, appelé GCS (Global Cut Seed) dans [7] et pattern dans [6]. Mais il peut également être réduit à l'ensemble des décisions de la branche menant de la racine à ce nœud [13]. Dans notre cas, nous enregistrons uniquement les domaines courants d'un sous-ensemble de variables (pour ρ^{red}, cela correspond aux variables ni u-éliminables ni r-éliminables) d'un réseau initial, ce qui nous permet de casser automatiquement certaines formes de symétries locales. De manière intéressante, on peut imaginer une combinaison entre les deux approches, en utilisant les "nogoods" extraits par notre méthode avec la détection par dominance via un ensemble pré-établi de symétries.

Finalement nous discutons de la complémentarité entre la recherche basée sur les états et l'enregistrement de nogoods (e.g. voir [5]). Un état donné, correspondant à un sous-réseau déjà montré insatisfiable, représente sous une forme compacte un nombre exponentiel de nogoods. D'un autre coté, un nogood (minimal) représente un nombre exponentiel d'instantiations. La complémentarité de ces deux paradigmes apparaît dans leur capacité à éviter des recherches redondantes. En effet, un nogood donné évite (ou coupe) plusieurs états, tandis que un état donné coupe plusieurs instantiations partielles i.e. des instantiations menant à un même état.

8 Expérimentations

Pour montrer l'intérêt pratique de l'approche décrite dans ce papier, nous avons effectué des expérimentations sur des instances de la seconde compétition de solveurs CSP (`http://cpai.ucc.ie/06/Competition.html`) sur un PC Pentium IV 2.4GHz 1024MiB sous Linux. Nous avons utilisé l'algorithme MGAC (intégrant GAC3^{rm} [11]), et étudié l'impact de la recherche basée sur les états, notée SBS, avec différentes heuristiques de choix de variables. Les performances sont mesurées en termes de nombre de nœuds visités (nodes), de temps cpu en secondes (cpu) ainsi qu'en nombre de nœuds écartés par SBS (hits). Il est important de remarquer que la recherche basée sur les états peut être appliquée sur des contraintes définies aussi bien en extension qu'en intention (mais, en fonction de l'opérateur de réduction utilisé, gérer des contraintes globales peut nécessiter un traitement spécifique).

Nous avons implémenté l'algorithme 1, mais en ne considérant qu'un sous-ensemble des variables de U_{elim}, étant donné que déterminer les variables u-éliminables impliquées dans des contraintes non binaires croît exponentiellement avec l'arité des contraintes. Plus précisément, dans notre implémentation, l'opérateur ρ^{uni} (appelé par ρ^{red}) supprime uniquement les variables ayant un domaine singleton et impliquées dans des contraintes ayant au plus une variable dont le domaine n'est pas singleton. Un tel ensemble peut être calculé en temps linéaire. Sur des réseaux binaires, n'importe quelle variable ayant un domaine singleton est automatiquement supprimée par notre opérateur (voir proposition 4). La structure de données implémentant la table de transposition utilisée pour stocker les sous-réseaux est une table de hachage dont la clef est calculée en concaténant les couples (id, dom) où id est un entier unique associé à chaque variable et dom le domaine de la variable elle-même représenté par un vecteur de bits. Lorsqu'on ne cherche qu'une seule solution, aucune information supplémentaire n'a besoin d'être enregistrée dans la table, et la présence de la clef est suffisante pour écarter un nœud.

La table 1 présente les résultats obtenus sur les instances du problème des $pigeons$. On peut observer l'intérêt de SBS sur ce problème puisque de nombreux nœuds sont évités. Sur ce type d'instance, on peut noter qu'il est plus intéressant d'utiliser l'heuristique $brelaz$ (des résultats identiques sont obtenus avec $dom/ddeg$ [1]) que $dom/wdeg$ [2]. Ceci s'explique par le fait que $brelaz$ est plus proche de l'ordre lexicographique, qui est bien adapté à ce problème.

Dans la table 2, nous nous concentrons sur certaines instances réelles difficiles du problème d'allocation de fréquences radio (RLFAP pour Radio Link Frequency Assignment). Étant données $1,200$ secondes de temps cpu, même avec SBS, ces instances ne peuvent pas être résolues en utilisant $brelaz$ ou $dom/ddeg$. C'est pourquoi nous ne présentons que les résultats obtenus avec $dom/wdeg$. On peut noter une amélioration par un facteur environ égal à 4

		$brelaz$		$dom/wdeg$	
		$\neg SBS$	SBS	$\neg SBS$	SBS
Pigeons-11	cpu	265.48	2.33	272.73	6.35
	$nodes$	$4,421K$	$5,065$	$4,441K$	$61,010$
	$hits$	0	$4,008$	0	$40,014$
Pigeons-13	cpu	$timeout$	4.57	$timeout$	26.44
	$nodes$	–	$24,498$	–	$327K$
	$hits$	–	$20,350$	–	$245K$
Pigeons-15	cpu	$timeout$	12.66	$timeout$	81.58
	$nodes$	–	$115K$	–	$900K$
	$hits$	–	$98,124$	–	$728K$
Pigeons-18	cpu	$timeout$	116.19	$timeout$	$timeout$
	$nodes$	–	$1,114K$	–	–
	$hits$	–	$983K$	–	–

TAB. 1 – Coût de MAC avec et sans SBS sur des instances du problème des Pigeons

		$dom/wdeg$	
		$\neg SBS$	SBS
scen11-f8	cpu	14.84	15.74
	$nodes$	$15,045$	$13,858$
	$hits$	0	370
scen11-f7	cpu	57.68	15.36
	$nodes$	$113K$	$14,265$
	$hits$	0	919
scen11-f6	cpu	110.18	18.67
	$nodes$	$217K$	$18,938$
	$hits$	0	1252
scen11-f5	cpu	550.55	162.32
	$nodes$	$1,147K$	$257K$
	$hits$	0	$17,265$

TAB. 2 – Coût de MAC avec et sans SBS sur des instances RLFAP difficiles

des performances lorsqu'on utilise SBS. La table 3 montre finalement les résultats obtenus sur des instances binaires et non-binaires (les instances dubois et pret comportent des contraintes ternaires) pour lesquelles notre approche est efficace.

Les résultats peuvent être résumés comme suit. Lorsque SBS n'est pas efficace, le solveur est pénalisé d'environ 15%. Cependant, en analysant le comportement de SBS, on peut décider à tout moment de stopper son utilisation et de libérer la mémoire requise pour stocker les différents sous-réseaux : le temps cpu perdu par le solveur est alors borné par le temps défini grâce à cette analyse. Quand SBS est efficace, et c'est le cas sur certaines séries, l'amélioration peut être très significative que ce soit en temps cpu ou en nombre d'instances résolues.

Une question récurrente porte sur la quantité de mémoire nécessaire pour stocker ces différents états. Beaucoup d'espace mémoire peut être économisé, en particulier sur les graphes de contraintes peu denses, en supprimant les variables u-éliminables et r-éliminables. Par exemple, seulement $265MiB$ sont nécessaires pour mémoriser les $50,273$ sous-réseaux différents stockés au cours de la résolution (en 162 secondes) de l'instance $scen11$-$f5$. Cette instance est composée de 680 variables et comporte jusqu'à 39 valeurs par domaine.

		brelaz		*dom/ddeg*		*dom/wdeg*	
Instances		$\neg SBS$	SBS	$\neg SBS$	SBS	$\neg SBS$	SBS
composed-25-10-20-4-ext	*cpu*	179.84	4.27	2.82	2.6	2.7	2.57
	nodes (hits)	$1,771K$	$9,944\ (2,609)$	$1,644$	$784\ (24)$	262	$255\ (5)$
composed-25-10-20-9-ext	*cpu*	857.07	3.73	12.13	10.25	2.58	2.66
	nodes (hits)	$10M$	$7,935\ (1,738)$	$75,589$	$54,245\ (2,486)$	323	$323\ (0)$
dubois-21-ext	*cpu*	*timeout*	480.98	*timeout*	384.84	911.51	295.81
	nodes (hits)	–	$6,292K\ (2,097K)$	–	$4,194K\ (2,097K)$	$16M$	$3,496K\ (1,573K)$
dubois-22-ext	*cpu*	*timeout*	*timeout*	*timeout*	*timeout*	*timeout*	527.19
	nodes (hits)	–	–	–	–	–	$6,641K\ (3,146K)$
pret-60-25-ext	*cpu*	687.31	5.65	471.16	5.82	416.49	2.27
	nodes (hits)	$12M$	$55,842\ (13,188)$	$7,822K$	$47,890\ (13,188)$	$7,752K$	$4,080\ (1,384)$
pret-150-25-ext	*cpu*	*timeout*	*timeout*	*timeout*	*timeout*	*timeout*	10.34
	nodes (hits)	–	–	–	–	–	$97,967\ (37,457)$

TAB. 3 – Coût de MGAC avec et sans SBS sur des instances structurées

9 Conclusion

Dans ce papier, nous apportons la preuve de faisabilité, dans le domaine de la satisfaction de contraintes, de l'exploitation d'une technique bien connue en recherche heuristique, l'élagage à partir des tables de transposition. Ceci n'a pas été réalisé auparavant puisqu'en satisfaction de contraintes contrairement à la recherche heuristique, deux branches d'un arbre de recherche ne peuvent pas mener à un même état. Ceci nous a amené à définir des opérateurs de réduction, extrayant une information partielle à partir d'un nœud, suffisante pour détecter que des réseaux de contraintes similaires n'ont pas besoin d'être explorés.

Nous fournissons des éléments sur les aspects à la fois théorique et pratique de l'exploitation de ces opérateurs en terme d'équivalence entre les nœuds. Deux pistes immédiates à ce travail concernent la définition d'opérateurs de réduction plus puissants ainsi que l'exploitation des propriétés de dominance entre les nœuds. De nombreux liens avec le concept de symétrie doivent encore être étudiés, et nous pouvons espérer une fertilisation croisée intéressante entre SBS et les méthodes d'élimination de symétries.

10 Remerciements

Ce papier a été supporté par le CNRS et le projet ANR "Planevo" n^{o}JC05_41940.

Références

[1] C. Bessiere and J. Régin. MAC and combined heuristics : two reasons to forsake FC (and CBJ ?) on hard problems. In *Proceedings of CP'96*, pages 61–75, 1996.

[2] F. Boussemart, F. Hemery, C. Lecoutre, and L. Sais. Boosting systematic search by weighting constraints. In *Proceedings of ECAI'04*, pages 146–150, 2004.

[3] D. Cohen, P. Jeavons, C. Jefferson, K.E. Petrie, and B. Smith. Symmetry definitions for constraint satisfaction problems. *Constraints*, 11(2-3) :115–137, 2006.

[4] R. Debruyne and C. Bessiere. Domain filtering consistencies. *Journal of Artificial Intelligence Research*, 14 :205–230, 2001.

[5] R. Dechter. Enhancement schemes for constraint processing : backjumping, learning and cutset decomposition. *Artificial Intelligence*, 41 :273–312, 1990.

[6] T. Fahle, S. Schamberger, and M. Sellman. Symmetry breaking. In *Proceedings of CP'01*, pages 93–107, 2001.

[7] F. Focacci and M. Milano. Global cut framework for removing symmetries. In *Proceedings of CP'01*, pages 77–92, 2001.

[8] E.C. Freuder. Eliminating interchangeable values in constraint satisfaction problems. In *Proceedings of AAAI'91*, pages 227–233, 1991.

[9] R. Greenblatt, D. Eastlake, and S. Crocker. The Greenblatt chess program. In *Proc. Fall Joint Computer Conference*, pages 801–810, 1967.

[10] J. Hoffmann and B. Nebel. The FF planning system : Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, 14 :253–302, 2001.

[11] C. Lecoutre and F. Hemery. A study of residual supports in arc consistency. In *Proceedings of IJCAI'07*, pages 125–130, 2007.

[12] T.A. Marsland. Computer chess and search. In *Encyclopedia of Artificial Intelligence*, pages 224–241. J. Wiley & Sons, 1992.

[13] J.F. Puget. Symmetry breaking revisited. *Constraints*, 10(1) :23–46, 2005.

[14] A. Reinefeld and T. A. Marsland. Enhanced iterative-deepening search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(7) :701–710, 1994.

[15] D. Slate and L. Atkin. Chess 4.5 : The northwestern university chess program. In *Chess Skill in Man and Machine*, pages 82–118. Springer Verlag, 1977.

[16] V. Vidal. A lookahead strategy for heuristic search planning. In *Proceedings of ICAPS'04*, pages 150–159, 2004.

[17] A. L. Zobrist. A new hashing method with applications for game playing. Technical Report 88, Computer Sciences Dept., Univ. of Wisconsin. Reprinted in *Int. Computer Chess Association Journal* 13(2) :169–173 (1990)., 1970.