Traitement des CSP partiellement symétriques

Florent Verrroust, Nicolas Prcovic

Traitement des CSP partiellement symétriques

Florent Verroust et Nicolas Prcovic

ILOG - Sophia Antipolis, LSIS - Université Paul Cézanne - Aix-Marseille III
fverroust@ilog.fr, nicolas.prcovic@lsis.org

Résumé

De nombreux CSP contiennent un mélange de contraintes symétriques et asymétriques. Nous présentons une approche générale qui permet d'appliquer les méthodes d'élimination de symétries connues à la partie symétrique d'un CSP puis de chercher une solution au problème entier en intégrant postérieurement les contraintes asymétriques. Nous étudions aussi le cas particulier des problèmes d'optimisation où seule la fonction de coût à minimiser empêche les symétries. Nous montrons expérimentalement que dans ce contexte-là nous pouvons accélérer la résolution de certains problèmes.

Abstract

Many CSPs contain a combination of symmetrical and asymmetrical constraints. We present a global approach that allows to apply already known methods for breaking symmetries on the symmetrical part of a CSP and then to search for a global solution by integrating afterwards the asymmetrical constraints. Then, we focus on optimization problems where only the cost function is asymmetrical. We show experimentally that in this case we can speed up the search of some problems.

1 Introduction

Le traitement des symétries dans les problèmes de satisfaction de contraintes (CSP) est un sujet de plus en plus étudié depuis une bonne dizaine d'années. Lorsqu'un CSP contient de nombreuses symétries, éviter d'explorer les larges portions de l'espace de recherche qui contiennent des instanciations partielles équivalentes permet d'obtenir des gains de temps très importants. Le formalisme CSP autorise qu'une contrainte soit définie comme une relation quelconque entre plusieurs variables, plus précisément comme n'importe quel sous ensemble du produit cartésien de leurs domaines. En conséquence, très peu de contraintes sont susceptibles a priori d'asseoir des symétries. Cependant, les problèmes pratiques contenant des symétries sont loin d'être négligeables même s'ils sont tout de même minoritaires. En fait, il s'avère qu'un nombre important de CSP contiennent un mélange de contraintes symétriques (contraintes d'égalité, de différence, somme de plusieurs variables égale à une constante, etc) et de contraintes asymétriques. Ces CSP sont donc globalement asymétriques et ne peuvent bénéficier directement des nombreuses techniques de traitement de symétries proposées jusqu'à présent. L'objectif principal de cet article est de proposer un schéma général de traitement de ces CSP asymétriques contenant des problèmes partiels symétriques et d'étendre ainsi l'application des techniques existantes de traitement de symétries. Cette idée de séparer le traitement des contraintes symétriques de celui des contraintes asymétriques est apparue récemment dans [4] et dans [3].

En section 2, nous illustrons l'intérêt de cette approche à travers deux exemples de CSP simples. En section 3, nous rappelons plusieurs notions sur les groupes de symétries qui nous sont utiles pour présenter formellement notre schéma général de résolution en section 4. Puis, nous traitons en section 5 du cas plus spécifique d'un certain type de problèmes d'optimisation où notre méthode est susceptible d'être particulièrement efficace. Nous l'expérimentons sur deux types de problèmes en section 6.

2 Exemples introductifs

Nous allons commencer par donner une idée générale de notre schéma de résolution à travers deux exemples simples. Dans le premier, nous considérons un CSP P à 3 variables x, y et z, définies chacune sur le même domaine $\{1, 2, 3\}$. Le CSP ne contient que deux contraintes : $xyz = 6$ et $x + 2y + 3z = 10$. Ce problème n'a qu'une seule solution : $x = 3$, $y = 2$ et $z = 1$. Ce problème ne contient aucune symétrie. La

taille de l'espace de recherche à explorer est de $3^3 = 27$ combinaisons de valeurs.

Considérons maintenant le problème P', qui est le même problème que P, mais qui ne garde que la première contrainte : $xyz = 6$. Ce problème contient toutes les symétries de variable possibles, qui font qu'à partir d'une solution quelconque, on peut en obtenir une autre en permutant des variables. En l'occurrence, l'ensemble des 6 solutions de P' est $x = 1$, $y = 2$ et $z = 3$ ainsi que n'importe laquelle de ses permutations de variables (échange entre x et y, échange entre x et z ou compositions de ces deux échanges). Une méthode de recherche permettant l'élimination de symétries est capable de déterminer une solution très rapidement. Par exemple, les techniques ajoutant des contraintes brisant les symétries avant la résolution (par exemple [7]) permettraient de poser les contraintes $x < y$ et $y < z$ qui briseraient les symétries de façon à ce qu'on n'obtienne qu'une seule solution, dite canonique : $x = 1$, $y = 2$ et $z = 3$. Ici, à cause de l'ajout de contraintes brisant les symétries, une simple application de la cohérence d'arc permettrait d'ailleurs de réduire les domaines des variables à la valeur de la solution finale ($x < y$ permet d'éliminer 1 du domaine de y et 3 du domaine de x, puis $y < z$ permet d'éliminer 1 et 2 du domaine de z, etc).

Nous savons que l'ensemble des solutions de P est inclus dans celui de P'. Pour obtenir une solution de P, il suffit d'énumérer les solutions de P' et de vérifier si elles respectent la contrainte $x + 2y + 3z = 10$. Enumérer les solutions de P', c'est essayer toutes les permutations de variables de la solution canonique. Le gain en temps que nous attendons est basé sur le fait que l'espace de recherche de P n'est plus l'ensemble des combinaisons de valeurs des variables (ici, de taille $3^3 = 27$) mais l'ensemble des solutions de P' (ici, de taille $3! = 6$), qui est bien plus petit. Pour que notre méthode soit efficace, il faut que le temps de calcul des solutions canoniques de P' ajouté au temps d'énumération de leurs symétriques soit plus court qu'une résolution classique de P.

Notre deuxième exemple est une version simplifiée[1] du problème de Vellino [9]. L'objectif de ce problème est de remplir k conteneurs de différents types (bleu, rouge ou vert) avec une liste de composants de différents types (verre, plastique, fer, bois ou cuivre). Chaque conteneur peut accueillir c composants, quel que soit le type du conteneur ou du composant. Le placement des composants dans les conteneurs doit respecter un certain nombre de contraintes : tous les conteneurs n'acceptent pas tous les types de composants ou n'en acceptent qu'un nombre limité, certains composants ne peuvent pas être simultanément présents dans un conteneur, etc. On peut modéliser le problème sous la forme d'un CSP comme suit. Les variables sont, d'une part, t_1, ...t_k, de domaine $\{rouge, vert, bleu\}$, où t_i représente la couleur du i^e conteneur et, d'autre part, $c_{1,1}$, ..., $c_{1,c}$, $c_{2,1}$, ..., $c_{k,c}$, $k \times c$ variables de domaine $\{rien, verre, plastique, fer, bois, cuivre\}$, où $c_{i,j}$ représente le j^e composant présent dans le i^e conteneur. A part les contraintes explicites mentionnées plus haut, nous avons les contraintes de cardinalité suivantes : pour chaque type x de composants, le nombre total de composants requis doit être égal à r_x : $|\{c_{i,j} = \mathrm{x} : 1 \leq i \leq k, 1 \leq j \leq c\}| = r_x$.

Si nous ne considérons que ces contraintes de cardinalité, nous sommes face à un problème contenant beaucoup de symétries[2] puisqu'il n'y a plus de contraintes entre les types de conteneurs et les types de composants. Nous avons des symétries de valeurs : toute valeur d'une variable t_i peut être changée par n'importe quelle autre (la couleur des conteneurs est indifférente). Nous avons des symétries de variables : les variables $c_{i,j}$ sont permutables entre elles (l'emplacement d'un composant est indifférent). Nous pouvons donc résoudre très efficacement ce problème partiel symétrique en n'en générant qu'une seule solution canonique :

- on utilise une seule couleur pour tous les conteneurs.
- on impose un ordre total sur les variables $c_{i,j}$ et les types de composants ($verre < plastique < fer < bois < cuivre < rien$). On affecte alors les variables $c_{i,j}$ ainsi : on affecte les r_{verre} premières variables $c_{i,j}$ à $verre$, les $r_{plastique}$ suivantes à $plastique$, etc, et, une fois tous les composants placés, on affecte les variables restantes à $rien$.

Nous savons maintenant que si une solution à notre problème complet existe, ce sera une permutation de valeurs des variables t_i et une permutation des variables $c_{i,j}$. Nous pouvons donc énumérer ces permutations en vérifiant les contraintes de compatibilités entre types de conteneurs et types de composants. Notre espace de recherche est alors d'emblée réduit car le nombre de permutations de la solution canonique du problème partiel est bien inférieur au nombre de combinaisons de valeurs à affecter aux variables ($6^{k.c} \times 3^k$).

Avant de présenter le schéma de résolution dans un cadre général, nous allons rappeler quelques notions utiles sur les symétries, les groupes de symétries et leur exploitation.

[1] Dans le problème originel, chaque type de conteneur a une capacité qui lui est propre et la valeur de k n'est pas donnée, il faut la minimiser.

[2] Il y en a déjà un certain nombre si on garde toutes les contraintes.

3 Rappel de définitions

Les définitions et notations que nous rappelons ici sont usuelles et peuvent par exemple se retrouver dans [8]. Au sens mathématique, un *groupe* est un ensemble G structuré par une opération binaire $\circ$ portant sur les éléments de cet ensemble, qui respecte les propriétés de fermeture (le résultat de l'opération $\circ$ sur deux éléments de G est un élément de G), d'associativité, d'élément neutre (l'élément neutre est noté e) et d'inversion (tout élément de G a un inverse). H est un sous-groupe de G, ce qu'on note $H \leq G$, si H est un sous-ensemble de G muni de la même opération binaire $\circ$ et que H forme lui-même un groupe.

Une *permutation* est une bijection d'un ensemble sur lui-même. Nous représentons une permutation par un ensemble de *cycles* de la forme $(\omega_1\ \omega_2\ ...\ \omega_k)$ qui signifie que $\forall\omega_j, \omega_{j+1}$ est l'image de ω_j et que ω_1 est l'image de ω_k. Ex : étant donnée la permutation (1 2 4)(3 5), les images respectives de 1, 2, 3, 4 et 5 sont 2, 4, 5, 1 et 3. Nous pouvons appliquer des permutations à des ensembles ou des suites d'éléments. Ex : étant donnée la permutation (1 2 4)(3 5), l'image du couple (1,5) est (2,3), l'image de (1,2,3,4,5) est (2,4,5,1,3), l'image de l'ensemble {2, 3, 4} est {1, 4, 5}.

Informellement, une symétrie est une permutation des éléments d'un ensemble Ω qui préserve les relations portant sur les éléments de cet ensemble. L'ensemble des symétries de Ω forment un groupe G, l'opération binaire $\circ$ étant la composition (de symétries). On dit que G *agit* sur Ω. Pour éviter certaines ambiguïtés, on appelle *points* les éléments de Ω et on réserve habituellement le terme *élément* aux éléments de G. Si $\sigma \in$ G, et $\omega \in \Omega$, on note (de façon standard) ω^σ l'image du point ω par la symétrie σ. Plutôt que d'écrire $\sigma_1 \circ \sigma_2$ (avec σ_2 devant s'appliquer avant σ_1), on abrège par $\sigma_1\sigma_2$, car $\circ$ est la seule opération possible entre deux symétries.

Définition 1 *Microstructure d'un CSP*
La microstructure (Ω, A) d'un CSP est un (hyper)graphe dont chaque sommet de Ω correspond à une affectation d'une de ses variables avec une valeur de son domaine et dont chaque (hyper)arête de A correspond à un k-uplet de valeurs autorisé par une contrainte d'arité k.

Définition 2 *Symétrie de CSP*
Soit (Ω, A) la microstructure d'un CSP. Une symétrie de CSP est une permutation de Ω qui, appliqué à A, laisse A inchangé.

Les symétries que nous traitons sont à prendre dans le sens le plus général possible. Nous ne nous restreignons ni aux symétries de variables (conservation des valeurs d'une instanciation mais pas des variables) ni aux symétries de valeurs (conservation des variables d'une instanciation mais pas des valeurs) : appliquée à une instanciation (partielle ou complète), l'ensemble des valeurs et des variables peut changer. Cette définition correspond à la *symétrie syntaxique* de [1] ou *symétrie de contraintes* de [2]. Le groupe G des symétries du CSP agit sur Ω en préservant la relation A.

La figure 1 présente notamment une grille 4×4 de 16 points. Chaque point correspond à l'affectation d'une valeur à une variable mais nous n'avons pas besoin de savoir lesquels pour appliquer les méthodes que nous allons présenter. Ce pourrait être une grille du problème des 4 dames à placer sur un échiquier (et on aurait les points 1, 2, 3, 4, 5, etc correspondant aux affectations de variables $x_1 = 1$, $x_1 = 2$, $x_1 = 3$, $x_1 = 4$, $x_2 = 1$, etc) ou à n'importe quel autre CSP dont la somme des tailles des domaines est 16 (CSP avec uniquement deux variables de domaines de taille 4 ou huit variables booléennes, etc) et qui possèdent les mêmes symétries.

Définition 3 *Orbite*
L'orbite ω^G du point ω d'un ensemble Ω sur lequel agit un groupe G est $\omega^G = \{\omega^\sigma : \sigma \in G\}$, c'est-à-dire l'ensemble des images de ω par application d'une permutation de G. On peut aussi généraliser la notion d'orbite à un ensemble $\Delta \subseteq \Omega$: $\Delta^G = \{\{\omega^\sigma : \omega \in \Delta\} : \sigma \in G\}$.

Ex : dans l'exemple de la figure 1, 1^G = {1, 4, 13, 16}, 2^G = {2, 3, 5, 8, 9, 12, 14, 15}, $\{1, 2\}^G$ = {{1, 2}, {1, 5}, {3, 4}, {4, 8}, {9, 13}, {13, 14}, {12, 16}, {15, 16}}.

Dans le cadre de cet article, nous pourrons avoir affaire à l'orbite d'un sommet du graphe de la microstructure (c'est-à-dire d'une affectation de variable de CSP) ou à l'orbite d'une instanciation partielle ou d'une solution de CSP, qui impliquent plusieurs affectations de variables. Si I=$\{\omega_1, \omega_2, ..., \omega_i\}$ est une instanciation partielle ou complète des variables du CSP alors l'ensemble des instanciations symétriques I^G est $\{I^\sigma : \sigma \in G\}$, c'est-à-dire $\{\ \{\omega_1^\sigma, \omega_2^\sigma, ..., \omega_i^\sigma\} : \sigma \in G\}$. En particulier, si I est une solution du CSP alors I^G ne contient que des solutions (symétriques), tandis que si I n'est pas une solution alors I^G ne contient aucune solution.

Définition 4 *Générateurs de groupe*
Les générateurs *d'un groupe G constituent un sous-ensemble H des éléments de G qui permettent, en se composant un certain nombre de fois entre eux, d'obtenir tous les éléments de G. On le note G=<H>.*

Ex : le groupe des 8 symétries d'une grille (voir figure 1) peut être généré à partir de compositions des deux générateurs suivants : la réflexion verticale γ_1 et la réflexion diagonale γ_2. L'ensemble des 8 symétries peut s'exprimer ainsi : e (l'identité), γ_1, γ_2, $\gamma_2\gamma_1$, $\gamma_1\gamma_2$, $\gamma_2\gamma_1\gamma_2$, $\gamma_1\gamma_2\gamma_1$ et $\gamma_1\gamma_2\gamma_1\gamma_2$.

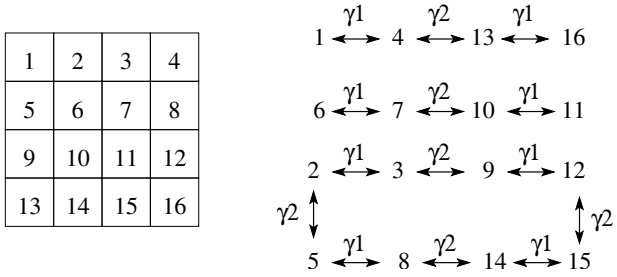

Figure 1: Une grille 4×4 et son graphe d'orbites. $\Omega = \{1, 2, ..., 16\}$. G=<{$\gamma_1$, γ_2}> avec γ_1 = {(1 4)(2 3)(5 8)(6 7)(9 12)(10 11)(13 16)(14 15)} et γ_2 = {(2 5)(3 9)(4 13)(7 10)(8 14)(12 15)}.

Une propriété importante de l'ensemble des générateurs est que son cardinal est borné par une fonction pseudolinéaire de $|\Omega|$. Ceci est à rapprocher du cardinal maximal d'un groupe, qui est $|\Omega|!$. Si nous obtenons la solution d'un CSP et que nous disposons explicitement du groupe de symétries du CSP, nous pouvons obtenir chacune des solutions symétriques en lui appliquant une fois chaque symétrie du groupe. Cependant, mémoriser chacune de ces symétries peut excéder les capacités d'une machine si le groupe est trop grand. Par contre, la taille modérée de l'ensemble des générateurs d'un groupe nous permet de mémoriser le groupe en compréhension et de générer toutes les solutions équivalentes en appliquant de toutes les manières possibles une succession de générateurs à la solution trouvée.

Définition 5 *Stabilisateur par point*
Le stabilisateur par point $G_{(\Delta)}$ de $\Delta \subseteq \Omega$ est le sous-groupe $G_{(\Delta)} = \{\sigma \in G : \forall \omega \in \Delta, \omega^\sigma = \omega\}$, c'est-à-dire l'ensemble des symétries de G qui laissent fixes tous les points de Δ.

Ex : dans l'exemple de la figure 1, G = <{γ_1, γ_2}>, $G_{(1)}$ = <{γ_2}>, $G_{(1,2)}$ = {e}.

Il est possible de représenter en compréhension un groupe de permutations G grâce à une base B=($\beta_1, ..., \beta_k$), qui est une suite de points de Ω. B est une base pour G si l'unique stabilisateur par point de B est l'identité, i.e : $G_{(B)}$ = {e}. La base B définit alors la chaîne de stabilisateurs par point suivante :

$$G=G^{[1]} \geq G^{[2]} \geq ... \geq G^{[k]} \geq G^{[k+1]} = \{e\}$$

où $G^{[j]} = G_{(\beta_1,...,\beta_{j-1})}$. Cette base permet de définir les éléments de G qui génèrent successivement $G^{[1]}$,...,$G^{[k+1]}$. Grâce par exemple à l'algorithme de Schreier-Sims [8], on peut construire un ensemble de générateurs $\{\gamma_i^{(j)} : 1 \leq j \leq k, 1 \leq i \leq t_j\}$ de G, appelé *ensemble de générateurs forts*, de telle façon que: $G_{(\beta_1,...,\beta_{h-1})} = <\{\gamma_i^{(j)} : h \leq j \leq k, 1 \leq i \leq t_j\}>$
En d'autres termes, les générateurs $\gamma_1^{(j)}, ..., \gamma_{t_j}^{(j)}$ laissent fixes les points β_1,... β_{h-1} mais pas le point β_h. Un outil tel que Nauty [5] permet d'obtenir une base, une chaîne de stabilisateurs et ses générateurs forts à partir d'un graphe. Nous pouvons ainsi obtenir automatiquement une représentation compacte du groupe de symétries d'une microstructure de CSP.

Définition 6 *Graphe d'orbites*
Un graphe d'orbites *est un graphe orienté dont les sommets sont les points de Ω. Quels que soient les sommets i et j, il existe un arc (i,j), et il est étiqueté γ, ssi $j = i^\gamma$ et γ est un générateur de G.*

Grâce à un graphe d'orbites, certaines questions sur les groupes de permutations peuvent se ramener à une question sur les graphes. Par exemple, les sommets d'une composante connexe d'un graphe d'orbites constituent une orbite (cf figure 1).

4 Schéma général de résolution

Considérons un CSP, appelé P, contenant n variables, dont les domaines sont discrets et dont les contraintes sont d'arité quelconque. On partitionne l'ensemble C des contraintes en deux ensembles C_{sym} et C_{reste}. Appelons P_{sym} le CSP qui correspond au problème P auquel on enlève les contraintes de C_{reste} pour ne conserver que celles de C_{sym}.

Pour que notre schéma de résolution ait un intérêt, il faut choisir C_{sym} de telle manière que P_{sym} contienne des symétries. En pratique, la plupart des contraintes ont leur sémantique et il est facile de savoir si elles induisent isolément des symétries. Par ailleurs, dans [6], des méthodes générales sont proposées pour agréger ces contraintes et en avoir une représentation sous forme de graphe dont on peut déterminer les symétries. Il y a donc beaucoup de cas pratiques où cette bipartition de contraintes est facile à déterminer, voire à automatiser.

La recherche d'une solution de P se fait d'une part en cherchant des solutions canoniques de P_{sym}, d'autre part en explorant l'orbite de chaque solution canonique de P_{sym} pour y trouver une instanciation qui respecte les contraintes de C_{reste}.

Pour résoudre P_{sym}, nous pouvons utiliser n'importe quelle méthode de traitement de symétries

connue. Il nous faut alors modifier l'algorithme de façon à ce que dès qu'une solution canonique I de P_{sym} est trouvée, on explore l'orbite de I pour trouver une instanciation qui respecte les contraintes de C_{reste}. S'il y en a une, on s'arrête car c'est une solution de P. Sinon, on continue la recherche de nouvelles solutions canoniques de P_{sym}. Ce schéma de résolution a été récemment proposé dans [3] mais il n'y est décrit aucun moyen concret et efficace de parcourir l'orbite. C'est ce que nous allons présenter maintenant. On peut envisager une exploration de l'orbite de I qui soit effectuée de manière systématique ou bien de manière incomplète (en prenant le risque de manquer une solution).

4.1 Recherche locale dans une orbite

Lorsque l'orbite d'une solution canonique de P_{sym} est grande, on peut envisager de n'en explorer qu'une partie grâce à une méthode de réparation locale. Or, nous pouvons très facilement adapter n'importe quelle méthode (Min-conflicts, recherche tabou, recuit simulé, etc) en considérant que le voisin d'une instanciation complète ne résulte pas de la modification de la valeur d'une seule variable mais de l'application d'un générateur. Ainsi, le voisinage d'une instanciation complète I est $\{ I^\gamma : \gamma \in H\}$ si G=<H>. L'évaluation de l'instanciation complète se fait en comptant le nombre de contraintes violées dans C_{reste}.

On peut guider la recherche grâce à une heuristique de choix de générateurs. La première heuristique venant naturellement à l'esprit est celle qui sélectionne en premier le générateur qui améliore le plus le coût de la solution. Une deuxième heuristique plus évoluée choisit parmi les générateurs améliorant la solution celui qui stabilise le plus de points et, en cas d'ex-aequo, celui améliorant le plus la solution. Une telle heuristique améliore la solution en apportant une modification la plus locale possible (celle portant sur les points non fixés), se rapprochant de ce que fait une recherche locale traditionnellement. La première heuristique est celle qui améliore le plus rapidement le coût mais elle est le susceptible d'atteindre rapidement un minimum local. Par contre, la seconde heuristisque avance plus prudemment en effectuant des modifications les plus indépendantes les unes des autres afin de continuer d'améliorer la solution le plus longtemps possible.

4.2 Exploration systématique d'une orbite

Si maintenant nous voulons énumérer toutes les solutions symétriques d'une orbite de solution canonique de P_{sym}, il faut être capable de retrouver toutes les symétries du groupe G de la microstructure de P_{sym} à partir de son ensemble de générateurs.

Une méthode couramment utilisée pour énumérer toutes les permutations de G est d'effectuer une recherche arborescente où à chaque nœud de l'arbre se trouve une permutation. On obtient tous les fils d'un nœud en appliquant chacun des générateurs à la permutation. On mémorise toutes les permutations au fur et à mesure qu'on les génère tout en vérifiant chacune pour savoir si on l'avait déjà découverte. Si c'est le cas, on abandonne la branche (on backtracke si c'est une recherche en profondeur d'abord). Le gros inconvénient de cette méthode est la complexité en mémoire, qui est celle de l'ordre (le nombre d'éléments) du groupe, dans le pire des cas égal à $|\Omega|!$. Il est nécessaire de mémoriser toutes les permutations pour la raison suivante. Chaque permutation de G peut s'exprimer comme étant la composée de générateurs de G sous la forme d'un mot $a_1 a_2 ... a_m$, où chaque a_i est soit un générateur, soit son inverse. Or, une même permutation peut s'exprimer sous la forme de plusieurs mots différents. Dans l'exemple de la figure 1, le demi-tour peut se représenter par les mots $\gamma_1\gamma_2\gamma_1\gamma_2$ ou $\gamma_2\gamma_1\gamma_2\gamma_1$. Il est facile et peu coûteux en mémoire de générer tous les mots d'une certaine longueur. A notre connaissance, il n'existe pas d'autre méthode générale permettant d'éviter de générer plusieurs fois la même permutation (sous la forme de mots tous différents) que celle qui calcule la permutation correspondant à chaque mot et maintient la liste de toutes les permutations déjà générées.

Comme l'orbite d'une solution est potentiellement trop grande pour être mémorisée, il nous faut quand même nous résigner à énumérer des mots, même si ça implique de générer plusieurs représentants d'une même symétrie. Cependant, notre but n'est pas de générer toutes les permutations d'une solution canonique de P_{sym} mais uniquement une permutation dont l'image est une solution de P (ie, qui respecte les contraintes de C_{reste}). Il nous faut en fait trouver un moyen de générer le moins possible de permutations.

Arbre de permutations

Considérons un groupe de permutations G, une base $(\beta_1, ..., \beta_k)$ de G et ses générateurs forts $\Gamma=\{\gamma_i^{(j)} : 1 \leq j \leq k,\ 1 \leq i \leq t_j\}$, où le groupe induit par $\{\gamma_1^i, ..., \gamma_{t_i}^i, ..., \gamma_1^k, ..., \gamma_{t_k}^k\}$ est $G^{[i]} = G_{(\beta_1,...,\beta_{i-1})}$, le stabilisateur de $(\beta_1, ..., \beta_{i-1})$. Toute permutation va bouger un certain nombre de points et en laisser certains fixes. Grâce à l'emboîtement des stabilisateurs déterminé par la base ainsi que du partitionnement de ses générateurs forts, nous pouvons toujours décomposer une permutation en plusieurs per-

mutations ne bougeant qu'une partie des points. Plus précisément, toute permutation peut s'exprimer sous la forme $\sigma_1\sigma_2...\sigma_k$, où chaque σ_i est une permutation (composée de plusieurs générateurs) qui bouge β_i et éventuellement d'autres points mais qui laisse fixes les points β_j, $\forall j < i$. Les permutations σ_i sont celles du stabilisateur $G_{(\beta_1,...\beta_{i-1})}$ qui bougent β_i. Toute permutation σ_i peut donc s'exprimer sous la forme de compositions de générateurs de l'ensemble $\{\gamma_h^{(j)} : i \leq j \leq k, 1 \leq h \leq t_j\}$. Enumérer l'orbite d'une solution canonique en lui appliquant toutes les permutations de G va alors consister à effectuer une recherche arborescente en profondeur d'abord. On part de la solution canonique et on lui applique toutes les permutations de la forme σ_1 (y compris l'identité), puis à chacune d'entre elles on applique toutes les permutations de la forme σ_2, etc, jusqu'à σ_k. L'ensemble des feuilles de l'arbre de recherche représente l'orbite de la solution canonique. Il nous reste maintenant à expliquer comment déterminer toutes les permutations de la forme σ_i.

Considérons le graphe d'orbites de G. Tout chemin dans ce graphe commençant par β_i est étiqueté par une suite de générateurs qui forment un mot qui, d'une part, correspond à une permutation qui bouge β_i, d'autre part, commence par un générateur appartenant à $\{\gamma_h^{(i)} : 1 \leq h \leq t_i\}$. L'ensemble de ces chemins correspond donc aux mots représentant les permutations qui bougent β_i. Si maintenant on retire du graphe d'orbites tous les arcs dont l'étiquette se retrouve sur un arc dont une extrêmité fait partie de l'ensemble $\{\beta_1, \beta_2, ... \beta_{i-1}\}$, l'ensemble des chemins commençant par β_i qui restent sont ceux correspondant aux permutations qui laissent fixes les points de $\{\beta_1, \beta_2, ... \beta_{i-1}\}$. Ils représentent les mots de la forme σ_i que nous cherchions.

La figure 2 montre l'arbre de recherche dans le cas où l'ensemble de points {1, 2, 14} est un ensemble de points dont on cherche l'orbite.

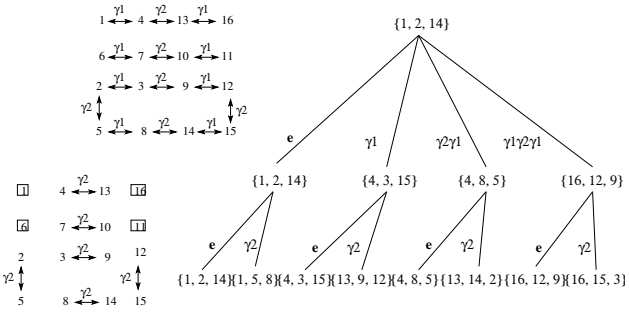

Figure 2: A droite, l'arbre de recherche générant l'orbite de {1, 2, 14}. A gauche, deux graphes d'orbites. Celui du haut est le graphe d'orbites de G et permet de générer les 4 débuts de permutations de la forme σ_1 du premier niveau de l'arbre. Celui du bas est le graphe d'orbites de $G_{(1)}$ et permet de générer les fins de permutations de la forme σ_2 du deuxième niveau. La feuille {4, 8, 5} permet de déterminer qu'en lui appliquant la permutation $\gamma_2\gamma_1 e = \gamma_2\gamma_1$ on retrouve la racine de l'arbre {1, 2, 14}.

Choix de la base

L'intérêt d'avoir un ensemble de générateurs forts construit à partir d'une base est que nous ne fixons pas tous les points à chaque fois que nous descendons d'un niveau dans l'arbre de recherche. Or, un point correspond à l'affectation d'une variable. Nous sommes donc sûrs que cette variable ne changera plus de valeur dans le sous-arbre de recherche où son point a été fixé. Remarquons qu'à chaque niveau de l'arbre de recherche, le fait de fixer un point entraîne souvent la fixation d'autres points. Sur le graphe d'orbites de $G_{(1)}$ de la figure 2, on peut constater que fixer le point 1 a aussi fixé les points 6, 11 et 16. La connaissance de l'ensemble des variables dont on sait qu'elles ne changeront plus de valeur nous permet de tester les contraintes de C_{reste} (ou d'appliquer des mécanismes de filtrage de domaines) et de backtracker en cas d'inconsistance sans avoir à explorer le sous-arbre de recherche. Backtracker en profondeur i nous permet de ne pas examiner le sous-arbre contenant toutes les permutations de la forme $\sigma_{i+1}...\sigma_k$ permettant de compléter la permutation $\sigma_1...\sigma_i$ à laquelle nous étions arrivés. La taille de la base est égale à la profondeur maximale de l'arbre, c'est-à-dire au nombre d'étapes où on peut tester certaines contraintes avant qu'une permutation de la forme $\sigma_1\sigma_2...$ soit complète. En conséquence, il faut faire en sorte de choisir la base qui contient le plus de points, car on a plus souvent la possibilité d'éliminer des sous-arbres correspondant à des complétions de permutations.

Filtrage

Par ailleurs, l'examen d'un graphe d'orbites permet de connaître l'ensemble des valeurs affectables à chaque variable, c'est-à-dire son domaine de valeurs possibles. L'union des orbites des points d'une solution représente les points qui peuvent apparaître dans les solutions symétriques. Certains points ne font partie d'aucune de ces orbites et on peut donc les éliminer des domaines des variables. Ex : si l'ensemble de points {1, 2, 14} (figure 2) est une solution d'un CSP alors les points 6, 7, 10 et 11 ne sont pas atteignables par une permutation et peuvent donc être éliminés des domaines. Ce filtrage des domaines peut

se compléter à chaque fois qu'on progresse dans la profondeur de l'arbre de recherche. En effet, le graphe d'orbites perd des arcs (puisque des points sont fixés) et de nouveaux points deviennent inatteignables. Ex : au nœud de profondeur 1 de l'arbre de recherche de la figure 2 contenant l'ensemble de points {1, 2, 14}, nous pouvons encore éliminer les points 4, 13, 3, 9, 12, 15 et 16 qui sont devenus inatteignables via les points 1, 2 ou 14. Evidemment, l'élimination d'une valeur dans un domaine par ce moyen peut permettre d'éliminer d'autres valeurs grâce aux contraintes de C_{reste} par application des techniques classiques de propagation (consistance d'arc, etc). De façon complémentaire, l'élimination d'une valeur par filtrage de contraintes de C_{reste} peut éliminer des points du graphe d'orbites et donc fixer d'autres points. Ex : si on considère le nœud de l'exemple précedent, éliminer par propagation de contraintes le point 5 du graphe d'orbites (du deuxième niveau) fixe le point 2. On ne peut donc plus appliquer de permutation à {1, 2, 14}, il devient inutile de développer le sous-arbre à partir de ce nœud. L'interaction entre le filtrage des domaines par examen du graphe d'orbites et par le maintien de formes de cohérences locales est complexe à cerner. Son étude dépasse le cadre de cet article mais méritera qu'on s'y attache si on veut rendre le plus efficace possible l'exploration d'une orbite de solution canonique de P_{sym}.

4.3 Complexité de la recherche dans une orbite

Le gain de temps que nous pouvons espérer est basé sur le fait que la taille de l'espace de recherche de P est supérieure ou égale à la somme des tailles de toutes les orbites de solutions canoniques de P_{sym}. En effet, ces orbites contiennent l'ensemble des solutions (canoniques ou non) de P_{sym}, qui est potentiellement beaucoup plus petit que l'espace de recherche de P (qui est l'ensemble des combinaisons de valeurs du problème).

Il n'est pas possible d'évaluer dans un cadre général le rapport de taille entre l'espace de recherche de P et la taille d'une orbite d'une solution canonique de P_{sym}. Mais nous pouvons le faire pour des cas particuliers. Nous allons le faire dans le cas où nous n'avons affaire qu'à des symétries de variables. Dans ce cas, les solutions symétriques de l'orbite contiendront les mêmes valeurs mais distribuées différemment dans les variables.

Un CSP P à n variables de domaines de taille d a un espace de recherche de d^n combinaisons de valeurs. Calculons maintenant la taille maximale de l'orbite d'une solution canonique de P_{sym}. Dans le pire des cas en terme de taille d'orbite, toute permutation de variables est une symétrie du groupe. Il y a m valeurs différentes dans la solution, avec $m \leq d$ et $m \leq n$. Nommons v_i le nombre d'occurrences de la i^e valeur dans une solution. La taille de l'orbite est $\mathrm{T}(n, m) = \frac{n!}{\prod_{1 \leq i \leq m} v_i!}$ (n! est le nombre de permutations de n éléments qu'il faut diviser par chaque v_i!, le nombre de permutations échangeant inutilement des valeurs identiques). Comme nous avons la relation $\sum_{1 \leq i \leq m} v_i = n$, on minimise le produit $\prod_{1 \leq i \leq n} v_i!$ quand les v_i ont les valeurs les plus proches possibles les unes des autres. Dans le pire des cas, toutes les valeurs v_i sont égales à $\frac{n}{m}$. Nous avons donc $\mathrm{T}(n, m) \leq \frac{n!}{((\frac{n}{m})!)^m}$. En approximant les factorielles grâce à la formule de Stirling $n! = \sqrt{2\pi} n^{n+\frac{1}{2}} e^{-n}(1+\epsilon(n))$ où $\epsilon(n)$ tend vers 0 quand n est grand, nous arrivons à $\mathrm{T}(n, m) \leq (\sqrt{2\pi})^{1-m} \frac{\sqrt{n}}{(\sqrt{\frac{n}{m}})^m} m^n \frac{1}{(1+\epsilon(n)^{m-1})}$. Nous avons donc $\mathrm{T}(n, m) \in O(\frac{m^{n+\frac{m}{2}}}{(2\pi)^{\frac{m}{2}} n^{\frac{m-1}{2}}})$. Comme il n'est pas évident de comparer cette complexité avec d^n, nous présentons en table 1 une comparaison des complexités en fonction de quelques valeurs de d, en prenant le cas le moins favorable où $m = d$.

d	d^n	$\mathrm{T}(n, m = d)$
2	2^n	$O(\frac{2^n}{\sqrt{n}})$
3	3^n	$O(\frac{3^n}{n})$
n	n^n	$O(\frac{\sqrt{n}}{(2\pi)^{\frac{n}{2}}} n^n)$

Table 1: Comparaison entre taille de l'espace de recherche d'un CSP et taille de l'orbite d'une solution dans le cas de symétries de variables. Par exemple, la ligne où $d = 3$ montre que si la complexité du calcul des solutions canoniques de P_{sym} est inférieur à $O(3^n)$ et que le nombre de ces solutions canoniques est inférieur à n alors la complexité de résolution de P est réduite grâce à notre méthode.

Pour avoir une comparaison globale entre une résolution classique et notre approche, il faut prendre en compte le temps de résolution de P_{sym}, qui reste a priori de l'ordre de $\Theta(d^n)$, et le fait que P_{sym} peut avoir un grand nombre de solutions canoniques et donc d'orbites à explorer. Notre approche ne peut donc être a priori efficace que si P_{sym} est rapide à résoudre et contient peu de solutions. Dans l'exemple du problème de Vellino que nous avons donné en section 2, P_{sym} possède une unique solution qui se calcule en temps linéaire. La taille de l'espace de recherche des variables $c_{i,j}$ est 6^n tandis que celle de l'orbite de l'unique solution est en $O(\frac{6^n}{n^{\frac{5}{2}}})$. Donc (sachant que rien ne change au niveau de l'énumération des variables $t_{i,j}$) nous avons divisé la complexité de la taille de l'espace de recherche par $n^{\frac{5}{2}}$.

5 Etude de cas : problème d'optimisation

Nous présentons maintenant un cas spécifique d'application de notre méthode où celle-ci a de bonnes chances d'être efficace. Un problème d'optimisation peut se définir par un CSP enrichi d'une fonction de coût portant sur les variables du CSP. L'objectif est de trouver la solution du problème qui minimise la fonction de coût. Ce type de problème peut se traiter en utilisant la méthode classique du Branch & Bound (B&B). On peut voir cette méthode comme effectuant la recherche d'une solution puis posant une contrainte de non dépassement du coût de la solution, réactualisée à chaque fois qu'une solution à un coût meilleur est trouvée. Si nous avons affaire à un CSP symétrique dont la fonction de coût brise les symétries, notre approche s'applique directement. Nous nous trouvons dans un cas particulier où c'est la méthode de résolution B&B elle-même qui applique une contrainte brisant les symétries après avoir trouvé une première solution. L'idée est donc de générer les solutions canoniques du CSP et, au fur et à mesure que ces solutions sont trouvées, de parcourir leur orbite à la recherche de la solution symétrique ayant le moindre coût. L'intérêt de cette méthode est que nous sommes à même de passer directement d'une solution à une autre sans avoir à continuer la résolution du CSP.

5.1 Version complète

Dans une version complète de notre méthode, nous devons parcourir l'orbite grâce à la recherche arborescente que nous avons décrite dans la section précédente. Pendant ce parcours d'orbite, il peut être possible de minorer et de majorer le coût de la meilleur solution symétrique de l'orbite, selon le type de la fonction de coût qui entre en jeu. Par exemple, si le coût est une fonction monotone ou linéaire de chacune des variables, le graphe d'orbites nous indique les plus petites et les plus grandes valeurs que peuvent prendre chaque variable et donc une borne minimale du coût. Si la borne minimale est supérieure au coût de la meilleure solution trouvée jusqu'à présent, il est inutile d'explorer l'orbite. La borne minimale est affinée lors de la descente dans une branche car des points (et donc des valeurs) sont fixés ou deviennent inatteignables, comme nous l'avons vu dans la section précédente.

5.2 Version locale recomplétée

Si un parcours complet de l'espace de recherche n'est pas envisageable à cause de la taille du problème, on peut envisager une méthode incomplète qui consiste en une recherche locale d'une meilleure solution dans les orbites. La recherche locale peut passer à côté de l'optimum de l'orbite mais elle peut permettre de trouver une solution à un coût suffisamment bon. Nous pouvons essayer de trouver une solution à un bon coût grâce un algorithme glouton : on essaie d'appliquer successivement chacun des générateurs jusqu'à ce qu'un d'eux produise une meilleure solution, et on itère ce processus jusqu'à aboutir à un optimum local.

Or, il s'avère que cette méthode peut facilement être rendue complète. Il suffit que le B&B qui génère les solutions ne soit plus soumis à des contraintes de canonicité et donc qu'il génère toutes les solutions y compris les symétriques. Dans cette version de la méthode, nous avons un B&B qui n'utilise aucune technique de traitement de symétrie mais qui, lorsqu'il trouve une nouvelle solution, va chercher de façon incomplète dans son orbite une solution à meilleur coût. Si une telle solution est trouvée, il va continuer la recherche où il s'était arrêté mais avec une borne de coût encore améliorée. Il va donc pouvoir élaguer encore plus l'arbre de recherche que s'il n'avait pas essayé d'explorer l'orbite. Le B&B étant une méthode complète, le fait de lui ajouter une procédure de recherche d'une meilleure borne conserve la complétude de la méthode.

6 Expérimentations et résultats

Le premier problème que nous expérimentons est le *carré magique pondéré*, qu'on trouve mentionné par exemple dans [4]. Le but est de remplir une grille $n \times n$ avec les entiers de 1 à n^2 de façon à ce que la somme de chaque ligne, de chaque colonne et de chacune des deux diagonales soit égale à $\frac{n(n^2+1)}{2}$. De plus, chaque case a un poids qui lui est propre et il faut minimiser la somme des produits du contenu des cases avec leur poids. La fonction coût est donc linéaire. Dans nos expérimentations, nous avons choisi chaque valeur de poids au hasard entre 1 et $100n^2$. L'ordre du groupe de symétries est 8 (ce sont les mêmes symétries que pour un problème de dames à placer sur un échiquier). Notre programme est écrit en utilisant Ilog Solver. A partir de la microstructure du CSP, on récupère une base de son groupe de permutation ainsi que ses générateurs forts grâce à Nauty [5].

Nous avons comparé trois méthodes de résolution : le B&B classique, le même B&B avec une exploration gloutonne d'orbites, que nous appelons *GreedySym*, et le B&B avec rechercherche arborescente systématique dans les orbites de solutions canoniques, que nous appelons *TreeSearchSym*. Pour chaque taille de prob-

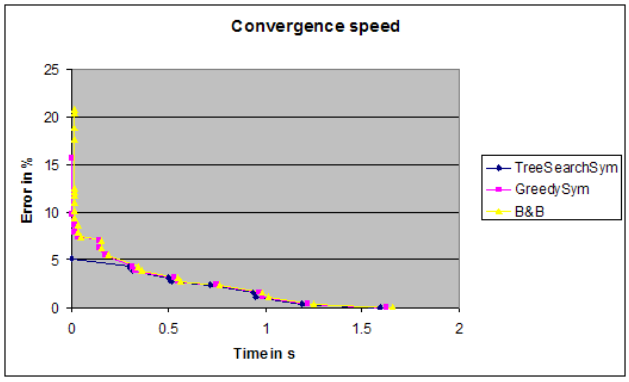

Figure 3: Vitesse de convergence vers la meilleure solution pour le problème du carré magique pondéré de taille 5. TreeSearchSym converge plus rapidement au début mais les trois courbes finissent pas coïncider et leurs temps de résolution sont égaux.

lème, nous avons exécuté 20 instances différentes et nous avons reporté le temps moyen (cf table 3 et 4). Nous pouvons constater que *TreeSearchSym* converge beaucoup plus rapidement que les deux autres méthodes vers une solution quasi-optimale pour $n = 5$. Pour $n = 6$, il trouve une solution à plus bas coût que les autres dans le temps imparti de 3 minutes qui avait été fixé.

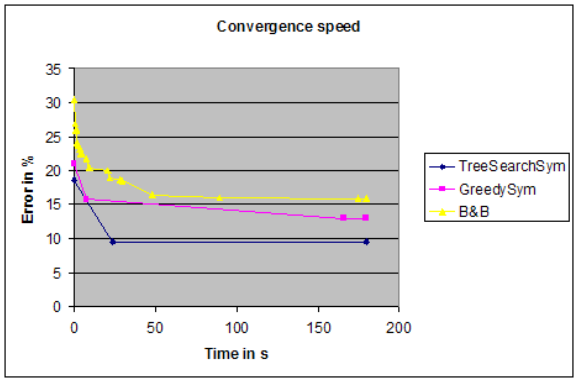

Figure 4: Vitesse de convergence vers la meilleure solution pour le problème du carré magique pondéré de taille 6. TreeSearchSym atteint plus rapidement une meilleure solution que GreedySearch and B&B à la fin des 3 minutes. Aucun ne termine la recherche au bout de plusieurs heures. (Finalement, l'outil de programmation mathématique CPLEX d'Ilog a été utilisé pour trouver la meilleure solution afin de nous permettre de connaître la distance entre celle-ci et celles que nos méthodes avaient trouvées.)

Le second problème choisi pour tester nos méthodes est un problème de coloriage de graphe. Le problème contient n (n est un multiple de 5) variables $\{x_1, ..., x_n\}$ à valeurs dans {0,1,2,3,4}. L'ensemble des contraintes est défini par, $\forall i$ tel que i est un multiple de 5 :

- $\{x_i, x_{i+1}, x_{i+2}, x_{i+3}\}$ ont des valeurs différentes.
- $\{x_i, x_{i+4}, x_{i+5}, x_{i+9}\}$ ont des valeurs différentes (sauf si i=n-5).

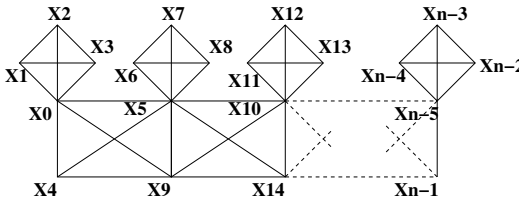

Figure 5: Graphe à colorier

Ce problème est décrit par la figure 5. Nous avons choisi ce problème car, bien qu'artificiel, il a l'avantage que son nombre de variables peut aisément être augmenté tout en conservant les mêmes symétries. En effet, quelque soit le nombre de variables présentes dans le modèle, il existe deux types de symétries de variable. Celles du premier type sont locales : $\forall$ i multiple de 5, les variables $\{x_{i+1}, x_{i+2}, x_{i+3}\}$ sont interchangeables. Il existe un deuxième type de symétrie qui est global. Considérons la partition de l'ensemble de variables en k parties de 5 variables définies par : $\forall j \in [1;k]\{x_j, x_{j+1}, x_{j+2}, x_{j+3}, x_{j+4}\}$. Ces ensembles de variables sont symétriques par la réflexion qui échange x_j et x_{k-j}. Cette fois-ci, l'ordre du groupe de symétries est une fonction exponentielle du nombre de variables.

La fonction coût est une fonction linéaire portant sur les valeurs prises par les variables. Chaque coefficient associé à une variable est un entier compris entre 1 et n. Pour éviter de résoudre continuellement le même problème, ces coefficients sont tirés aléatoirement de façon indépendante.

Nb variables	TreeSearchSym	GreedySym	B&B
15	0.054	0.0035	0.0042
20	0.25	0.025	0.026
25	2.35	0.16	0.17
30	11.6	1.04	1.11
35	94.6	6.39	6.7
40	492	44.5	47.3

Table 2: Temps de résolution (en secondes) du problème de coloriage de graphe.

La comparaison entre les trois techniques de résolution (B&B, GreedySym et TreeSearchSym) a été ef-

fectuée avec 50 résolutions de problèmes de taille identique dont la fonction coût changeait aléatoirement à chaque fois. Les résultats sont indiqués en table 2 et en figure 6.

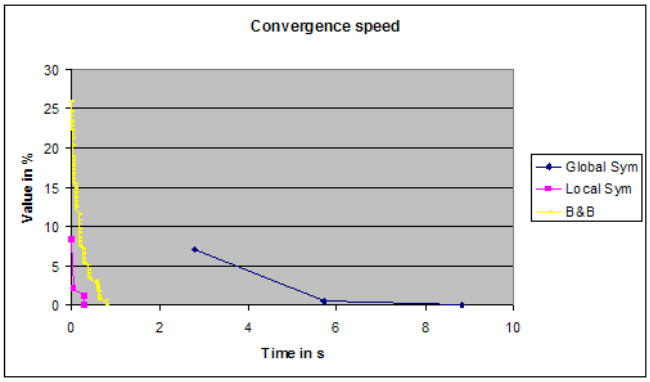

Figure 6: Vitesse de convergence vers la meilleure solution pour le problème de coloriage de graphe pour $n = 30$. GreedySym converge plus rapidement que B&B. TreeSearchSym converge très lentement. Le même comportement a été observé pour les autres valeurs de n.

Les temps de résolution avec GreedySym ne sont qu'un peu plus rapide (<10%) que celle avec le B&B. Cependant, GreedySym atteint la solution optimale bien avant B&B. TreeSearchSym a de mauvaises résutlats parce qu'il prend beaucoup de temps à fouiller systématiquement de grandes orbites.

7 Conclusion et perspectives

Grâce à une idée simple, nous avons étendu le champ d'application des méthodes d'élimination de symétries: lorsqu'un problème ne contient pas de symétries parce que quelques contraintes sont asymétriques, ce qui constitue un cas beaucoup plus courant que celui où les problèmes sont symétriques, nous pouvons quand même appliquer ces méthodes en laissant les contraintes gênantes momentanément de côté pour les réintégrer plus tard et terminer de résoudre le problème. Cela implique qu'on sache explorer efficacement l'orbite d'une solution canonique. Nous avons vu qu'une exploration incomplète de l'orbite par réparation locale est facile à mettre en œuvre. Par contre, une exploration systématique par recherche arborescente est plus délicate et nécessite que soient étudiés plusieurs aspects fondamentaux (choix de la base, adaptation des techniques de filtrage de domaines) pour qu'elle soit le plus efficace possible. Notre étude de ces aspects reste embryonnaire et devra être poursuivie. De plus, nous nous sommes intéressés à une classe spécifique de problèmes qui entraient dans le cadre général de notre approche : les problèmes d'optimisation où seule la fonction de coût est asymétrique. Dans ce contexte, nous avons proposé une autre méthode complète basée sur l'exploration locale d'orbites de solution. Comme elle peut sauter très rapidement d'une solution à une autre lors de l'exploration d'une orbite, elle permet d'améliorer la borne de coût plus rapidement. Il nous reste à vérifier l'efficacité de notre démarche dans le cadre plus général dans lequel nous nous sommes placés au départ : celui des problèmes de décision contenant un nombre conséquent de contraintes asymétriques. Ce type de problème est très courant en pratique et il serait très utile que nous perfectionnions notre approche suffisamment pour qu'elle reste efficace dans ce contexte.

References

[1] B. Benhamou. Study of symmetry in constraint satisfaction problems. In *Second Workshop on Principles and Practice of Constraint Programming (PPCP'94)*, 1994.

[2] D. Cohen, P. Jevons, C. Jefferson, K. E. Petrie, and B. Smith. Symmetry definitions for constraint satisfaction problems. In *Proceedings of CP'05*, pages 17–31, 2005.

[3] W. Harvey. Symmetric relaxation techniques for constraint programming. In *SymNet Workshop on Almost-Symmetry in Search*, pages 20–59, 2005.

[4] R. Martin. Approaches to symmetry breaking for weak symmetries. In *SymNet Workshop on Almost-Symmetry in Search*, pages 37–49, 2005.

[5] Brendan D. McKay. Practical graph isomorphism. *Congressus Numerantium*, 30:45–87, 1981.

[6] Jean François Puget. Automatic detection of variable and value symmetries. In *Proceedings of CP'05*, pages 475–489, 2005.

[7] Jean François Puget. Breaking all value symmetries in surjection problems. In *Proceedings of CP'05*, pages 490–504, 2005.

[8] Ako Seress. *Permutation Group Algorithms*. Cambridge University Press, 1999.

[9] Vellino. http://www.cs.york.ac.uk/aig/constraints/automodel/essence/pdf_models/prob116.pdf.