Elimination des symétries pour l'appariement de graphes

Stéphane Zampelli, Yves Deville, Pierre Dupont

Elimination des symétries pour l'appariement de graphes

Zampelli Stéphane, Yves Deville et Pierre Dupont

Université catholique de Louvain,
Département d'Ingénierie Informatique
2, Place Sainte-Barbe
1348 Louvain-la-Neuve (Belgique)
{sz,yde,pdupont}@info.ucl.ac.be

Résumé

L'appariement de graphes est une technique utilisée dans de nombreux domaines et peut être modélisé comme un problème de satisfaction de contraintes. Cette approche par contraintes est concurrentielle avec des algorithmes dédiés. Dans cet article, nous développons des techniques de détection et d'élimination de symétries spécifiques pour l'appariement de graphes. Nous montrons également comment ces symétries peuvent être éliminées. Nous montrons aussi comment les symétries de valeur conditionnelles peuvent être automatiquement détectées et utilisées. Une nouveau type de symétrie est présenté, appelé symétrie locale, et nous indiquons comment ce nouveau type de symétrie peut être calculé et exploité. Enfin, nous évaluons les techniques classiques d'élimination de symétries de variable et de valeurs sur des instances difficiles.

1 Introduction

Une symétrie dans un problème de satisfaction de contraintes est une fonction bijective qui transforme des solutions en solutions. Les symétries sont importantes car elles créent des sous-arbres symétriques dans l'arbre de recherche. S'il n'y a pas de solutions, l'absence de solutions doit être prouvée plusieurs fois dans des sous-arbres symétriques. Si des solutions existent, beaucoup de solutions symétriques seront énumérées dans des sous-arbres également symétriques. La détection et l'élimination des symétries permet par conséquent d'éliminer ces sous-arbres symétriques et d'accélérer la résolution des problèmes de satisfaction de contraintes.

Les symétries sont présentes naturellement dans les graphes puisqu'une permutation peut être vue comme un automorphisme dans un graphe. Les symétries d'un graphe peuvent être calculées au moyen d'algorithmes comme NAUTY [17]. Cependant, bien que plusieurs problèmes de graphe aient été traités [1, 2, 25] et qu'un domaine de calcul pour les graphes ait été défini [8], et malgré le fait que les symétries et les graphes sont reliés, l'utilisation de la détection et de l'élimination de symétries pour les problèmes de graphe n'a pas été étudié en programmation par contraintes.

Cet article vise à appliquer et à étendre les techniques de détection et d'élimination des symétries à l'appariement de graphes. Nous montrons que l'état de l'art des techniques de symétrie appliqué à l'appariement des graphes est insuffisant. Les techniques existantes utilisent en général seulement les symétries initiales, et ne détectent pas les symétries dynamiques qui se créent durant la recherche. Ces symétries dynamiques sont appelées symétries conditionnelles [10]. Nous montrons comment détecter et utiliser ces symétries conditionnelles.

Etat de l'art L'élimination des symétries est un thème de recherche connu dans le domaine de la programmation par contraintes. Crawford et al. [6] ont montré que le calcul d'un ensemble de prédicats éliminant toutes les symétries est un problème NP-hard. Diverses approches existent pour exploiter les symétries. Les symétries peuvent être éliminées durant la recherche soit en postant des contraintes additionnelles (SBDS) [14, 12] soit en élaguant l'arbre en-dessous d'un état symétrique à un état précédant (SBDD) [13]. Les symétries peuvent être également éliminées en les prenant en compte dans l'heuristique [18]. L'idée principale est de sélectionner la variable incluse dans le plus grand nombre de symétries

dans l'état courant, de telle sorte que les symétries sont éliminées dès que possible par l'heuristique. Les symétries peuvent être éliminées en ajoutant des contraintes au problème initial au noeud racine [6, 11]. Les symétries peuvent également être éliminées en remodélisant le problème [26].

Plus récemment, des efforts de recherche ont porté sur la définition, la détection et l'élimination de symétries. Cohen et al. [4] ont défini deux types de symétries, des symétries de solution et des symétries de contrainte. Ils ont prouvé que l'ensemble des symétries de contrainte est un sous-ensemble des solutions de symétries. De plus, Gent et al. [10] ont introduit la notion de symétrie conditionnelle et ont évalué plusieurs techniques pour éliminer ce type de symétries. Des symétries conditionnelles sont des symétries qui se créent durant la recherche. Cependant la détection de symétries conditionnelles restent un sujet de recherche. Les symétries peuvent être utilisées pour produire des types de consistances plus fortes et des mécanismes plus efficaces pour les établir [9]. Finalement, Puget [22] a montré comment détecter des symétries automatiquement, et a montré que toutes les symétries de variable peuvent être éliminées avec un nombre linéaire de contraintes dans les problèmes injectifs [21]. Pour les problèmes surjectifs, Puget [20] montre qu'ajouter une variable par valeur du problème plus un nombre linéaire de contraintes binaires permet d'éliminer toutes les symétries de valeur.

L'appariement de graphes est une technique centrale dans beaucoup de domaines d'application [5]. Différents types d'algorithmes ont été proposés, depuis des méthodes générales jusqu'à des algorithmes spécifiques pour certaines classes particulières de graphe. En programmation par contraintes, plusieurs auteurs [15, 24] ont montré que l'appariement de graphes peut être formulé comme un problème de programmation par contrainte, et ont suggéré que la programmation par contraintes pourrait être un outil puissant pour faire face à sa complexité combinatoire. En particulier, un modèle pour l'isomorphisme de sous-graphe a été proposé par Rudolf [24] et Valiente et al. [15]. Notre modélisation [28] est basée sur ces travaux. Une vision déclarative de l'isomorphisme de sous-graphe a aussi été proposé dans [16]. Dans [28], nous avons montré que l'approche par contraintes est concurrentielle avec des algorithmes dédiés sur une base de données représentant des graphes de différentes topologies.

Objectifs Notre travail vise à développer des techniques d'élimination de symétries pour l'appariement de graphes modélisé comme un problème de satisfaction de contraintes. Notre objectif premier est de développer des techniques de détection spécifiques pour l'élimination des symétries de variable et de valeur. Notre second objectif est de développer des techniques de symétries avancées qui peuvent être détectées dans le cas de l'appariement de graphes.

Résultats

- Nous montrons que les symétries de variable et de valeur peuvent être détectées en calculant l'ensemble des automorphismes du graph source et du graphe cible.
- Nous montrons que les symétries de valeur conditionnelles peuvent être détectées en calculant l'ensemble des automorphismes sur différents sous-graphes dans le graphe cible, appelés sous-graphes cibles dynamiques. La méthode GE-Tree peut être étendue pour exploiter ces symétries.
- Nous présentons le concept de symétries de valeur locales, qui sont des symétries de valeur sur un sous-problème. Nous montrons comment ces nouvelles symétries peuvent être calculées et exploitées en utilisant des méthodes standards tel que GE-Tree.
- Nos résultats expérimentaux comparent et analysent les améliorations atteintes et montrent que l'élimination de symétries non dynamiques permet d'augmenter le nombre d'instances tractables, mais que cette augmentation n'est pas significative.

Plan La section 2 définit la notion d'appariement de graphes et donne les définitions essentielles concernant l'élimination des symétries. La section 3 décrit une approche par contraintes pour l'appariement de graphes. La section 3 et 4 présentent des symétries de variable et des symétries de valeur pour l'appariement de graphes. Les symétries de valeur conditionnelles sont traitées en section 6, et la section 7 introduit les symétries de valeur locales. Enfin, la section 8 décrit nos résultats expérimentaux et la section 9 conclut cet article.

2 Définitions

2.1 Appariement de graphes

Avant de présenter le modèle de base pour l'appariement de graphes, nous définissons cette notion.

Un **graphe** $G = (N, E)$ est constitué d'un **ensemble de noeuds** N et d'un **ensemble d'arcs** $E \subseteq N \times N$, où un arc (u, v) est un couple de noeuds. Les noeuds u et v sont les extrémités de l'arc (u, v). Nous considérons des graphes orientés et non orientés.

Un **sous-graphe** d'un graphe $G = (N, E)$ est un graphe $S = (N', E')$ où N' est un sous-ensemble de N et E' est un sous-ensemble de E.

Un **isomorphisme de sous-graphe** entre un graphe source $G_p = (N_p, E_p)$ et un graphe cible $G_t = (N_t, E_t)$ est une fonction totale $f : N_p \rightarrow N_t$ respectant ces deux conditions :

1. la fonction f est injective

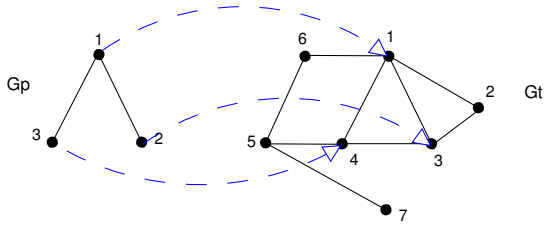

Figure 1: Example de solution pour une instance du problème du monomorphisme de sous-graphe.

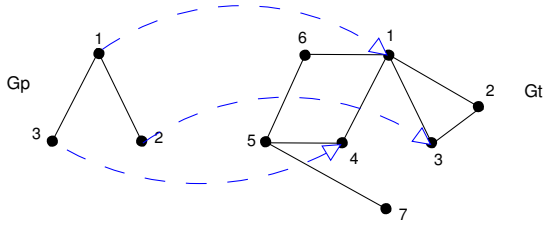

Figure 2: Example de solution pour une instance du problème de l'isomorphisme de sous-graphe.

2. f est un isormophisme : $(u, v) \in E_p \Leftrightarrow (f(u), f(v)) \in E_t$.

Un **monomorphisme de sous-graphe** entre G_p et G_t est une fonction totale $f : N_p \rightarrow N_t$ respectant ces deux conditions :

1. la fonction f est injective
2. f est un monomorphisme : $(u, v) \in E_p \Rightarrow (f(u), f(v)) \in E_t$.

Un **appariement de graphes** est soit un isomorphisme de sous-graphe soit un monomorphisme de sous-graphe.

La **fonction de voisinage** $V : N \rightarrow N$ est définie comme $V(i) = \{j \mid (i, j) \in E\}$. Nous notons V_p pour la fonction de voisinage du graphe source et V_t pour la fonction de voisinage du graphe cible.

Dans cet article, nous nous concentrons sur le monomorphisme de sous-graphe. Les résultats exposés ici restent valables pour l'isomorphisme de sous-graphe.

2.2 Symétries

Une instance d'un problème de satisfaction de contraintes est un triplet $< X, D, C >$ où X est l'ensemble des variables, D est le domaine universel spécifiant les valeurs possibles pour ces variables, et C est l'ensemble des contraintes. Dans le reste de ce document, $n = |N_p|$, $d = |D|$, et $D(x_i)$ est le domaine des x_i.

Une symétrie d'une instance P est une bijection σ des solutions vers les solutions, et des non solutions vers les non solutions [22].

Puisqu'une symétrie est une bijection où le domaine et le codomaine sont les mêmes, une symétrie est une permutation. Par exemple, la permutation $(a_1a_2)(b_1b_2b_3)$ est une bijection $\sigma(a_1) = a_2$, $\sigma(a_2) = a_1$, $\sigma(b_1) = b_2$, $\sigma(b_2) = b_3$, $\sigma(b_3) = b_1$ et $\sigma(c) = c$ sinon.

Une **symétrie de variable** est une fonction bijective $\sigma : X \rightarrow X$ permutant une (non) solution $s = ((x_1, d_1), \ldots, (x_n, d_n))$ vers une (non) solution $s' = ((\sigma(x_1), d_1), \ldots, (\sigma(x_n), d_n))$. Par exemple, la contrainte $x + y = 5$ implique la symétrie de variable $(x\ y)$.

Une **symétrie de valeur** est une fonction bijective $\sigma : D \rightarrow D$ permutant une (non) solution $s = ((x_1, d_1), \ldots, (x_n, d_n))$ vers une (non) solution $s' = ((x_1, \sigma(d_1)), \ldots, (x_n, \sigma(d_n))$. Par exemple, la contrainte x mod $3 = 2$ implies la symétrie de valeur $(8\ 5)$.

Une **symétrie de variable et de valeur** est une fonction bijective $\sigma : X \times D \rightarrow X \times D$ permutant une (non) solution $s = ((x_1, d_1), \ldots, (x_n, d_n))$ vers une (non) solution
$s' = (\sigma(x_1, d_1), \ldots, \sigma(x_n, d_n))$. Par exemple, considérons le problème de satisfaction de contraintes $D(x) = [1, 2, 4]$, $D(y) = [1, 3, 4]$, $D(z) = [4]$, $x + y = 5$, $y \leq z$. L'ensemble des solutions est $\{(x, 1),(y, 4),(z, 4)\}$, $\{(x, 4),(y, 1),(z, 4))$, $((x, 2), (y, 3), (z, 4))\}$. Une symétrie de valeur et de variable est $((x, 1)\ (x, 4))$, $((y, 4)\ (y, 1))$. Notez que $(x\ y)$ n'est pas une symétrie de variable et $(1\ 4)$ n'est pas une symétrie de valeur.

Une **symétrie conditionnelle** d'un problème P est une symétrie contenant un sous-problème $P^{'}$ de P. Les conditions d'une symétrie conditionnelle sont les contraintes nécessaires pour générer $P^{'}$ à partir de P [10].

Un **groupe** est un ensemble finit et infini d'éléments avec une opération binaire (appelé l'opération du groupe) qui satisfait les quatre propriétés : la propriété de cloture, la propriété d'associativité, la propriété d'identité, et la propriété d'inverse. Un **automorphisme de graphe** est un isomorphisme de graphe avec lui-même. L'ensemble des automorphismes $Aut(G)$ définit un groupe de permutation fini.

Exploiter les symétries consistent en trois étapes : détection de symétries, élimination des symétries pour réduire l'espace de recherche, et génération de toutes les solutions.

2.3 But de l'élimination des symétries

Le but général de l'élimination des symétries est de trouver un sous-ensemble de solutions canoniques [19].

Sans perte de généralité, nous pouvons appliquer un ordre arbitraire sur les variables et les valeurs. Soit $\leq_{lex}$ un ordre sur les vecteurs représentant les solutions. Etant donné G le groupe de symétrie et Sol

l'ensemble des solutions, le sous-ensemble $BSol$ de solutions canoniques est défini comme :

$$BSol = \{s \in Sol \mid s \leq_{lex} \sigma(s) \ \ \forall \sigma \in G\}.$$

Les solutions Sol peuvent être générées en appliquant les éléments de G à $BSol$:

$$Sol = \{\sigma(s) \mid \sigma \in G \wedge s \in Bsol\}.$$

3 Approche par contraintes pour le monomorphisme de sous-graphe

Le modèle du monomorphisme de sous-graphe doit representer une fonction totale $f : N_p \rightarrow N_t$. Cette fonction totale peut être modélisé avec $X = x_1, ..., x_n$ où x_i représente le noeud numéro i de G_p et $D = N_t$. L'ensemble des variables est l'ensemble des noeuds sources et leur domaine initial est l'ensemble des noeuds cibles.

La contrainte injective peut être déclarée en utilisant `alldiff` $(x_1, ..., x_n)$.

La contrainte de monomorphisme déclare que si un arc existe entre deux noeuds sources, alors un arc doit exister entre leur image respective :

$$\forall\ (i, j) \in E_p : (f(i), f(j)) \in E_t\ .$$

Pour chaque $(i, j) \in E_p$, la contrainte de monomorphisme de base est définie comme :

$$MC(x_i, x_j) \equiv (x_i, x_j) \in E_t\ .$$

Une contrainte globale $MC(x_1, ..., x_n)$ peut être formulée, au lieu d'avoir une contrainte MC par paire de noeuds :

$$MC(x_1, ..., x_n) = \bigwedge_{(i,j)\in E_p} MC(x_i, x_j)\ .$$

Une contrainte redondante élaguant l'espace de recherche a été proposé dans [15]. Cette contrainte réduit le temps de recherche pour des instances difficiles. Cette contrainte redondante est une contrainte Alldiff locale [23] sur le voisinage d'un noeud, en remarquant que le nombre de candidats disponibles dans l'union des domaines des voisins x_i ne peut pas être inférieure au nombre des voisins x_i dans le graphe source :

$$LA(x_i) \equiv |\cup_{j\in V_p(i)} D(x_j) \cap V_t(x_i)| \geq |V_p(i)|\ .$$

Une contrainte algorithmique globale $LA(x_1, ..., x_n)$ peut être formulée :

$$LA(x_1, ..., x_n) \equiv \bigwedge_i LA(x_i)\ .$$

Les contraintes pour le problème du monomorphisme de graphe sont les suivantes :

`alldiff`$(x_1, ...x_n)$, `MC`$(x_1, ..., x_n)$ et `LA`$(x_1, ..., x_n)$.

L'implémentation, une comparaison avec des algorithmes dédicacés, et une extension à l'isomorphisme de sous-graphe peut être trouvé dans [28]. L'extension de ce cadre pour le matching approximatif utilisant un domaine de calcul pour les graphes et les fonctions a été présenté dans [7].

4 Symétries de variable

4.1 Détection

Cette section montre que, dans l'appariement de graphes, les symétries de variables sont les automorphismes du graphe source et ne dépendent pas du graphe cible.

Comme expliqué dans [22], l'ensemble des symétries de variable d'un problème de satisfaction de contraintes est le groupe d'automorphisme d'un *graphe symbolique*. Chaque contrainte est transformée en un graphe symbolique. Le groupe d'automorphisme de ce graphe symbolique est l'ensemble des symétries de la contrainte. Le graphe symbolique final est obtenu en fusionant les noeuds qui jouent le même rôle dans les graphes symboliques. Le groupe d'automorphisme peut être calculé en utilisant des outils comme NAUTY [17]. Ces outils renvoient un ensemble de générateurs du groupe utile pour éliminer les symétries.

Nous allons appliquer ces idées au graphe source, qui représente le graphe symbolique du réseau de contraintes. Le graphe source G_p est transformé en un graphe symbolique $S(G_p)$, où $Aut(S(G_p))$ est l'ensemble des symétries de variable.

Définition 1 *Un problème de satisfaction de contraintes P modélisant une instance du monomorphisme de sous-graphe (G_p, G_t) peut être transformé dans le graphe symbolique $S(P)$ suivant :*

1. *Chaque variable x_i est un noeud distinct labélisé var*
2. *S'il existe une contrainte $MC(x_i, x_j)$, alors il existe un arc entre i et j dans le graphe symbolique*
3. *La contrainte alldiff, comme suggéré dans [22], est transformé en un noeud typé avec le label 'a'; un arc (a, x_i) est ajouté au graphe symbolique pour chaque x_i.*

Parce que les contraintes LA sont redondantes, elles ne modifient pas l'ensemble des solutions, et donc elles ne modifient pas l'ensemble des symétries de variables de P. La contrainte LA peut donc être omise dans le graphe symbolique.

Si nous ne considérons pas le noeud supplémentaire et les arcs introduits par la contrainte alldiff, alors le graphe symbolique $S(P)$ et G_p sont isomorphes par construction.

Figure 3: Example de graphe symbolique pour un graphe source carré.

Etant donné les labels des noeuds, un automorphisme dans $S(P)$ envoie le noeud alldiff avec lui-même et les noeuds correspondant aux variables à un autre noeud correspondant à une variable. Chaque automorphisme dans $Aut(G_p)$ est donc une restriction d'un automorphisme $Aut(S(P))$, et un élément dans $Aut(S(P))$ est une extension d'un élement dans $Aut(G_p)$. Les deux théorèmes suivants en découlent :

Théorème 1 *Etant donné un monomorphisme de sous-graphe (G_p, G_t) et le problème de satisfaction de contrainte correspondant P :*

- $\forall\, \sigma \in Aut(G_p)\ \exists\, \sigma^{'} \in Aut(S(P)) :$
  $\forall\, n \in N_p : \sigma(n) = \sigma^{'}(n)$
- $\forall\, \sigma^{'} \in Aut(S(P))\ \exists\, \sigma \in Aut(G_p) :$
  $\forall\, n \in N_p : \sigma(n) = \sigma^{'}(n)$

Théorème 2 *Etant donné un monomorphisme de sous-graphe (G_p, G_t) et le problème de satisfaction de contrainte correpondant P, l'ensemble des symétries de variable de P est l'ensemble des fonctions bijectives $Aut(S(P))$ limitées à N_p, qui est égal à $Aut(G_p)$.*

Le théorème 2 affirme que seul $Aut(G_p)$ doit être calculé pour obtenir toutes les symétries de variables.

La figure 3 montre un graphe source transformé en graphe symbolique.

La figure 4 donne quelques examples de symétries de variable. Pour chaque graphe source, la liste des genrateurs et la taille du groupe sont donnés. Le graphe triangle non orienté a deux générateurs $(2\ 3)$ et $(1\ 2)$ et 3! automorphismes (avec e la fonction identité) :

1. $\sigma_1 = \begin{pmatrix} 2 & 3 \end{pmatrix}$
2. $\sigma_2 = \begin{pmatrix} 1 & 2 \end{pmatrix}$
3. $\sigma_1 \cdot \sigma_2 = \sigma_1\sigma_2 = \begin{pmatrix} 1 & 3 & 2 \end{pmatrix}$
4. $\sigma_2 \cdot \sigma_1 = \sigma_2\sigma_1 = \begin{pmatrix} 1 & 2 & 3 \end{pmatrix}$
5. $\sigma_1 \cdot \sigma_2\sigma_1 = \sigma_1\sigma_2\sigma_1 = e$
6. $\sigma_2 \cdot \sigma_1\sigma_2 = \sigma_2\sigma_1\sigma_2 = \begin{pmatrix} 1 & 3 \end{pmatrix}$

Le groupe d'automorphisme du graphe carré non orienté, connu comme D_4, a deux générateurs et huit automorphismes. Le triangle orienté a deux générateurs et trois automorphismes : $\{\begin{pmatrix} 1 & 2 & 3 \end{pmatrix}, \begin{pmatrix} 1 & 3 & 2 \end{pmatrix}, e\}$.

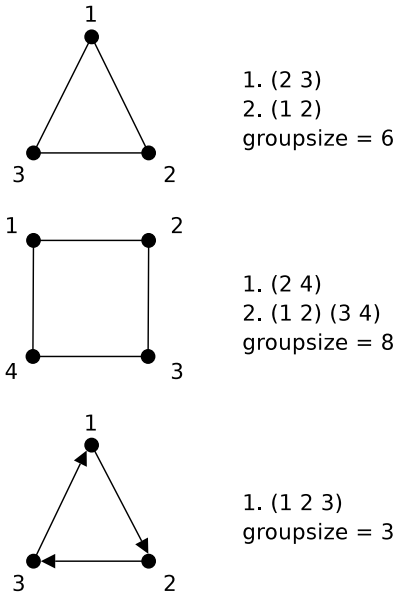

Figure 4: Example de graphes sources avec leurs générateurs.

4.2 Elimination

Deux techniques ont été sélectionnées pour éliminer les symétries. La première technique est une approximation et consiste à éliminer seulement les générateurs du groupe de symétrie [6]. Ces générateurs sont obtenus en utilisant un logiciel comme NAUTY, qui renvoie les générateurs du groupe de symétrie. Pour chaque symétrie de variable σ, une contrainte d'ordre est postée pour garder seulement les solutions canoniques. Puisque $s \leq \sigma s \Leftrightarrow ((x_1, v_1), \cdots, (x_n, v_n)) \leq ((\sigma(x_1), v_1), \cdots, (\sigma(x_n), v_n))$, une contrainte $x_1 \leq \sigma(x_1)$ est postée pour respecter l'ordre lexicographique.

La seconde technique élimine toute les symétries de variable d'un problème injectif en utilisant l'algorithm SchreierSims, sous l'hypothèse que les générateurs du groupe de symétrie de variable sont connus [22]. Dans un problème injectif comme celui de l'isomorphisme de sous-graphe, Puget a montré que le nombre de contraintes à poster est linéaire avec le nombre de variables. L'algorithme Scherier-Sims est une méthode efficace pour calculer une base et un ensemble générateur d'un groupe de permitation. Cet algorithme prend les générateurs en entrée et s'exécute en $O(n^2 log^3|G| + t.n.log|G|)$ où G est le groupe, t le nombre de générateurs et n la taille du groupe de toutes les permutations contenant G. L'ensemble générateur renvoyé est précisément l'information requise pour poster les contraintes d'élimination de symétrie.

Ces deux techniques seront comparées dans la section expérimentale.

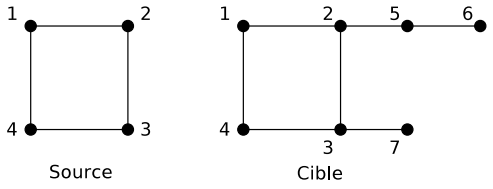

Figure 5: Example d'appariement où l'ensemble des symétries de valeur n'est pas vide et $Aut(G_t) = \emptyset$.

5 Symétries de valeur

5.1 Détection

Les symétries de valeur sont des automorphismes du graphe cible et ne dépendent pas du graphe source.

Théorème 3 *Etant donné un monomorphisme de sous-graphe (G_p, G_t) et son problème de satisfaction de contraintes P, chaque $\sigma \in Aut(G_t)$ est une symétrie de valeur de P.*

Preuve Supposons que $Sol = (v_1, \cdots, v_n)$ soit une solution. Considérez le sous-graphe $G = (N, E)$ de G_t, où $N = \{v_1, \cdots, v_n\}$ and $E = \{(i,j) \mid (\sigma^{-1}(i), \sigma^{-1}(j)) \in E_p\}$. Ceci signifie qu'il existe une fonction monomorphique $f^{'}$ envoyant G_p sur σG. Donc $((x_1, \sigma(v_1)), \cdots, (x_n, \sigma(v_n)))$ est une solution.

■

Toutes les symétries de valeur de P ne sont pas dans $Aut(G_t)$. Considérez la figure 5. Il existe deux solutions symétriques : $\{(x_1,1),(x_2,2),(x_3,3),(x_4,4)\}$ et $\{(x_1,2),(x_2,1),(x_3,4),(x_4,3)\}$ bien que $Aut(G_t) = \emptyset$.

La figure 6 donne un exemple de symétrie de valeur sur le graphe cible. Il y a seulement un générateur pour ce graphe : $(1\ 2)$. Supposons que le graphe source soit un chemin de lonueur 2 : $x_1 \rightarrow x_2 \rightarrow x_3$. Supposons que $(1,3,2)$ soit une solution. Alors $(2,3,1)$ est aussi une solution. Supposons que $(1,3,4)$ est une solutions. Alors $(2,3,4)$ est aussi une solution.

5.2 Elimination

Eliminer les symétries de valeur initiales peut être effectué en utilisant la méthode GE-Tree [3]. L'idée est de modifier la distribution en évitant les affectations de valeurs symétriques. Supponsons qu'un état S est atteint, où $x_1, \cdots, x_k$ sont assignés à $v_1, \cdots, v_k$ respectivement, et $x_{k+1}, \cdots, x_n$ ne sont pas encore assignés. La variable x_{k+1} ne doit pas être assignée à deux valeurs symétriques, puisque deux sous-arbres symétriques seraient parcourus. Pour chaque valeur $v_i \in D(x_{k+1})$ qui est symétrique à une valeur $v_j \in D(x_{k+1})$, seul un état S_1 devrait être généré avec la nouvelle contrainte $x_{k+1} = v_i$; aucun nouvel état S_2 avec $x_i = v_j$ ne devrait être généré.

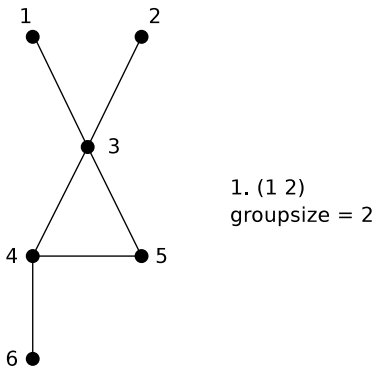

Figure 6: Example d'une symétrie de valeur sur le graphe cible.

Une manière directe pour calculer ces valeurs symétriques est de calculer une base et un ensemble générateur au moyen de SchrierSims. L'algorithme SchreierSims renvoie les sous-groupes de $Aut(G_t)$ G_i $(1 \leq i \leq d)$ tel que $\forall\ \sigma \in G_i : \sigma(j) = j\ \forall\ j \in [1,i]$. De plus SchreierSims renvoie l'ensemble des images de i qui laissent $0, \cdots, i$ invariants : $U_{i+1} = (i+1)^{G_{i+1}}$. Ces ensembles U_i sont intéressants parce qu'ils donnent l'ensemble des valeurs symétriques de $i+1$ étant donné que les valeurs $1, ..., i$ ne sont sujets à aucune permutation.

Pour utiliser ces U_i, les valeurs sont assignées dans un ordre croissant, si bien que l'hypothèse que $1, \cdots, i$ n'est sujet à aucune permitation est assurée. Supposons qu'un état S est atteint, $x_1, \cdots, x_k$ sont assignés à $v_1, \cdots, v_k$ respectivement, avec $v_1 \leq \cdots \leq v_k$ and $v_i \leq v_j\ \forall\ i \in [1,k]\ \forall\ j \in [k+1,d]$. Les variables $x_{k+1}, \cdots, x_n$ ne sont pas encore assignées. La prochaine valeur $v_{k+1} \in D(x_j)$ est sélectionnée dans l'ordre croisssant et est assignée à x_j. Deux nouveaux états S_1 and S_2 sont créés. La contrainte $x_{k+1} = v_{k+1}$ est incluse dans S_1 et les contraintes $x_{k+1} \neq v_{k+1}$ and $x_{k+1} \neq k^{'}\ \forall\ k^{'} \in U_k$. Les symétries de valeur dans l'état S_2 ont été effacées pour x_{k+1}.

6 Symétries de valeur conditionnelles

Dans le monomorphisme de sous-graphe, les relations entre les valeurs sont explicitement représentées sont le graphe cible. Cette propriété permet la détection de symétries de valeur conditionnelles.

6.1 Détection

Pendant la recherche, le graph cible perd un noeud a lorsque $a \notin \cup_{i \in N_p} D(x_i)$. Ceci est intéressant

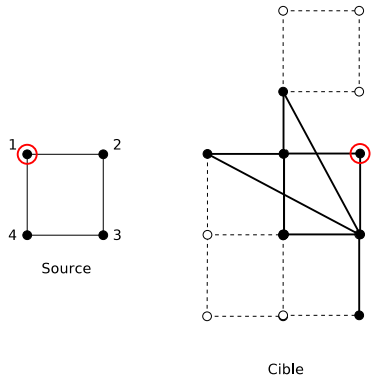

Figure 7: Example de sous-graphe cible dynamique.

parce que la relation entre les valeurs est connue dynamiquement.

L'ensemble des valeurs $\cup_{i \in N_p} D(x_i)$ sont les noeuds du sous-graphe G_t dans lequel on recherche une solution. Etant donné un état S, ce sous-graphe dynamique peut être calculé efficacement.

Nous définissons tout d'abord le sous-graphe dynamique de G_t.

Définition 2 *Soit S un état durant la recherche où $x_1, \cdots, x_k$ sont assignés, et $x_{k+1}, \cdots, x_n$ ne sont pas assignés. Le* ***graphe cible dynamique*** *$G_t^* = (N_t^*, E_t^*)$ est un sous-graphe de G_t tel que :*

- $N_t^* = \cup_{i \in [1, \cdots, n]} D(x_i)$
- $E_t^* = \{(a, b) \in E_t \mid a \in N_t^* \wedge b \in N_t^*\}$

La figure 7 montre un example de graphe cible dynamique. Dans cette figure, les noeuds encerclés sont assignés ensemble. Les noeuds blancs sont les noeuds exclus de $\cup_{i \in [1, \cdots, n]} D(x_i)$, et les noeuds noirs sont les noeuds inclus dans $\cup_{i \in [1, \cdots, n]} D(x_i)$. Les arc plains sont les arcs sélectionnés pour le sous-graphe cible dynamique.

Chaque automorphisme de G_t^* est une symétrie de valeur conditionnelle pour l'état S.

Théorème 4 *Etant donné un monomorphisme de sous-graphe (G_p, G_t), son problème de satisfaction de contraintes P, et un état S durant la recherche, chaque $\sigma \in Aut(G_t^*)$ est une symétrie de valeur conditionnelle de P. De plus, les conditions de σ sont $x_1 = v_1, \cdots, x_k = v_k$.*

Preuve Supposons $Sol = (v_1, \cdots, v_k)$ est une solution partielle. Considérons le sous-graphe G_t^*. L'état S peut être considéré comme un nouveau CSP P' d'une instance (G_p, G_t^*) avec des contraintes additionelles $x_1 = v_1, \cdots, x_k = v_k$. Par le théorème 3, la thèse suit.

■

La taille de G_t^* est un problème important, puisque nous l'utiliserons pour calculer les symétries conditionnelles. Le théorème suivant montre que, à cause des contraintes MC, le plus long chemin dans G_p a la même longueur que le plus long chemin dans G_t^* dès qu'une variable est assignée.

Définition 3 *Soit $G = (N, E)$ un graphe. Alors $maxd(G)$ représente la taille du plus long chemin simple entre deux noeuds $a, b \in N$.*

Théorème 5 *Etant donné un monomorphisme de sous-graphe (G_p, G_t), son problème de satisfaction de contraintes P, et un état S durant la recherche, si $\exists\, i \in N_p \mid |D(x_i)| = 1$, alors $maxd(G_p) = maxd(G_t^*)$.*

Ceci est un résultat intéressant en ce qui concerne les problèmes de complexité, lorsque $maxd(G_p)$ est petit. Dans la figure 7, $maxd(G_p)$=2 et seulement les noeuds à distance deux du noeud 1 dans le graphe cible sont inclus dans G_t^*.

Le graphe cible dynamique G peut être calculé dynamiquement. Dans [7], nous avons montré comment le monomorphisme de sous-graphe peut être modélisé et implémenté dans CP(Graph), un nouveau domaine de calcul pour les graphes. Dans ce cadre, une variable graphe T est utilisé comme graphe cible, avec le domaine initial $[\emptyset, \cdots, G_t]$. Lorsqu'une solution est trouvée, T est instantié au sous-graphe solution de G_t. Par conséquent le graphe target dynamique G_t^* sera la borne supérieure de la variable T et peut être obtenu en $O(1)$.

6.2 Elimination

Nous montrons comment modifier la méthode GE-Tree pour utiliser les symétries conditionnelles de valeur. Avant le labeling des variables, les actions suivantes sont effectuées :

1. Obtenir G_t^*.
2. Les algorithmes NAUTY et SchreierSims sont appelés. De cette étape les ensembles $U_i^{'}$ sont obtenus.
3. Le problème principal est comment adapter la sélection des variables et des valeurs de telle sorte que les symétries conditionnelles de valeur soient éliminées. Dans la méthode GE-Tree, à partir d'un état S, deux branches sont créées :
   (a) un nouvel état S_1 avec une contrainte $x_k = v_k$
   (b) un nouvel état S_2 avec les contraintes :
      i. $x_k \neq v_k$
      ii. $x_k \neq v_j \; \forall \; j \in U_{k-1}$.

   Pour utiliser les symétries conditionnelles de valeur, nous modifions légèrement cette méthode. Pour un état donné S, deux branches sont créées :
   (a) un nouvel état S_1 avec une contrainte $x_k = v_k$

(b) un nouvel état S_2 avec les contraintes :

i. $x_k \neq v_k$

ii. $x_k \neq v_j \; \forall \; j \in U_{k-1} \cup U_{k-1}^{'}$

Un problème est de savoir comment maintenir la structure U. Dans le système Gecode (http://www.gecode.org), dans lequel l'implémentation a été faite, les états sont copiés et il est inutile de maintenir explicitement la structure U. Donc la structure U ne doit pas être mise à jour à cause du backtracking. Une seule copie globale est gardée durant tout le processus de recherche.

Dans un état S où les symétries conditionelles sont découvertes, la structure U est copié dans une nouvelle structure $U^{''}$ et fusionné avec $U^{'}$. Cette structure $U^{''}$ sera utilisé pour tous les états $S^{'}$ ayant S comme prédecesseur.

Bien sur, une heuristique doit être ajouté pour choisir les états où une nouvelle symétrie de valeur conditionelle devrait être calculée.

7 Symétries de valeur locales

Dans cette section, nous présentons le concept de symétrie de valeur locale, c'est-à-dire de symétrie de valeur sur un sous-problème. De telles symétries seront détectées et exploitées durant la recherche.

7.1 Détection

Nous présentons d'abord les graphes dynamiques partiels. Ces graphes sont associés à un état durant la recherche et correspondent à la partie non résolue du problème. Nous pouvons les voir comme un nouveau problème local à l'état courant.

Définition 4 *Soit S un état durant la recherche dont les variables $x_1, \cdots, x_k$ sont assignées à $v_1, \cdots, v_k$ respectivement, et $x_{k+1}, \cdots, x_n$ ne sont pas encore assignés.*
*Le **graphe source partiel dynamique** $G_p^- = (N_p^-, E_p^-)$ est un sous-graphe de G_p tel que :*

- $N_p^- = \{i \in [k+1, d]\}$
- $E_p^- = \{(a,b) \in E_p \mid a \in N_p^- \wedge b \in N_p^-\}$

*Le **graph cible partiel dynamique** $G_t^- = (N_t^-, E_t^-)$ est un sous-graphe de G_t tel que :*

- $N_t^- = \cup_{i \in [k+1,d]} D(x_i)$
- $E_t^- = \{(a,b) \in E_t \mid a \in N_t^- \wedge b \in N_t^-\}$

Lorsque du forward checking (FC) est utilisé durant la recherche, dans n'importe quel état dans l'arbre de recherche, chaque contrainte comprenant *une* variable non instantiée est arc consistant. Autrement dit, chaque valeur dans le domaine d'une variable non instantiée est consistante avec une solution partielle. Cette propiété de FC sur un problème de satisfaction de contraintes binaire assure que l'on peut se focaliser sur les variables non instantiées et les contraintes associées sans perdre ou créer des solutions au problème initial. Cette propriété est vraie également lorsque la recherche atteint des formes de consistance plus fortes durant la recherche (Partial Look Ahead, Maintaining Arc Consistency, ...).

Théorème 6 *Soit (G_p, G_t) une instance du monomorphisme de sous-graphe, P son problème de satisfaction de contraintes associé, et S un état de P durant la recherche, où les variables assignées sont $x_1, \cdots, x_k$ avec les valeurs $v_1, \cdots, v_k$. Soit P' un nouveau problème de satisfaction de contraintes d'une instance du monomorphisme de sous-graphe (G_p^-, G_t^-) avec des contraintes additionnelles $x_{k+1}^{'} = D(x_{k+1}), \cdots, x_n^{'} = D(x_n)$. Alors :*

1. *Chaque $\sigma \in Aut(G_t^-)$ est une symétrie de valeur de $P^{'}$.*
2. *Si nous avons la propriété FC, nous avons $((x_1, v_1), \cdots, (x_n, v_n)) \in Sol(S)$ si et seulement si $((x_{k+1}^{'}, v_{k+1}), \cdots, (x_n^{'}, v_n)) \in Sol(P^{'})$.*

Ce théorème dit que les symétries de valeur du CSP local P' peuvent être obtenues en calculant $Aut(G_t^-)$, et que ces symétries peuvent être exploitées sans perdre ou ajouter de solutions au problème initial.

Il est important de remarquer que les symétries de valeur de P' ne sont *pas* des symétries conditionelles de P. Il n'est pas possible en effet d'ajouter des contraintes à P pour générer P'. Puisque le problème P' est un problème local associé à un état S, ces symétries de valeur sont appelées *symétries de valeur locales*.

Le graphe G_t^- peut être facilement calculé grâce aux variables graphes. Si T est la variable représentant le graphe cible sur le domaine initial $[\emptyset, \cdots, G_t]$, alors durant la recherche G_t^- est $lub(T) \setminus glb(T)$.

Considérons le monomorphisme de sous-graphe (G_p, G_t) dans la figure 8. Les noeuds du graphe source sont les variables du problème correspondant, i.e. le noeud i de G_p correspond à la variable x_i. Supposons que x_1 ait été assigné à la valeur 1. Parce que MC(x_1, x_3), $D(x_3) = \{4, 6, 7\}$. De plus, à cause de alldiff$(x_1, \cdots, x_n)$, la valeur 1 est effacé de tous les domaines $D(x_i)$ $(i \neq 1)$. Le nouveau problème $P^{'}$ constitué du sous-graphe de $G_p^- = (\{2, 3, 4, 5\}, \{(2, 3), (3, 2), (3, 5), (5, 3), (4, 5), (5, 4), (2, 4), (4, 2)\})$ et $G_t^- = (\{2, 3, 4, 5, 6, 7\}, \{(2, 3), (3, 2), (3, 5), (5, 3), (4, 5), (5, 4), (2, 4), (4, 2), (6, 7), (7, 6)\})$. Les domaines des variables de $P^{'}$ sont : $D(x_3) = \{4, 6, 7\}$, $D(x_2) = \{2, 5, 6, 7\}$, $D(x_5) = \{2, 5, 6, 7\}$, $D(x_3) = \{3, 4, 6, 7\}$. Pour l'état S, $Sol(S) = \{(1, 5, 4, 3, 2), (1, 2, 4, 3, 5)\}$ et $BSol(S) = \{(1, 2, 4, 3, 5)\}$. Pour le sous-problème $P^{'}$, $Sol(P^{'}) = \{(5, 4, 3, 2), (2, 4, 3, 5)\}$ et $BSol(P^{'}) = \{(2, 4, 3, 5)\}$. L'assignement partiel $(x_1, 1)$ dans l'état S avec les solutions de $P^{'}$ égale $Sol(S)$.

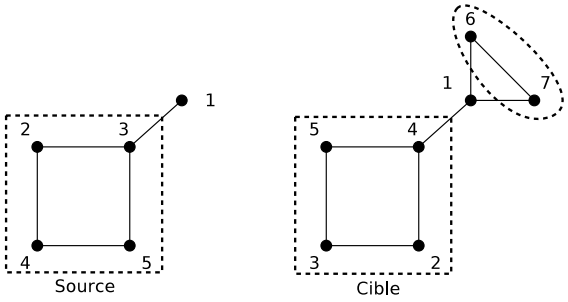

Figure 8: Example de symmétries de valeur locales. Les pointillés montrent le nouveau monomorphisme de sous-graphe du problème P'.

7.2 Elimination

L'élimination des symétries de valeur locales est équivalent à l'élimination des symétries de valeur sur le sous-problème P'. Les méthodes de Puget et celle du GE-Tree dynamique peuvent donc être utilisées dans le cas du problème local P'.

8 Résultats expérimentaux

Le modèle pour le monomorphisme de sous-graphe a été implémenté dans Gecode (http://www.gecode.org), en utilisant CP(Graph) et CP(Map) [8, 7]. CP(Graph) propose des variables dont le domaine représente un graphe et CP(Map) propose des variables dont le domaine représente une fonction. L'implémentation standard de NAUTY a été utilisée. Nous avons également implémenté l'algorithme SchreierSims. Le calcul des contraintes pour les problèmes injectifs a été implémenté, et la méthode GE-Tree a également été implémentée.

Nous avons évalué la détection et l'élimination des symétries de variable et de valeur, et la combinaison de ces deux techniques.

Les graphes utilisés pour générer les instances proviennent de la base de donnée GraphBase qui contient des graphes avec des topologies diverses. Cette base de graphes a été utilisée dans [15]. Il y a un ensemble de graphes orientés et un ensemble de graphes non orientés. Nous avons sélectionné les 30 premiers graphes pour les instances orientées et les 50 premiers graphes pour les instances non orientées. L'ensemble des graphes varient de 10 à 138 noeuds pour l'ensemble non orienté, et de 10 à 462 noeuds pour l'ensemble non orienté. En utilisant ces graphes, il y a 405 instances pour les graphes orientés et 1225 instances pour les graphes non orientés. Toutes les exécutions ont été effectuées sur un processeur dual Intel(R) Xeon(TM) CPU 2.66GHz avec 2 Go de mémoire vive.

Un problème important est de savoir combien de temps est nécesaire pour le preprocessing des graphes. NAUTY traite chaque graphe non orienté en moins de 0.02 secondes. Pour les graphes orientés, chaque graphe est traité en moins de 0.01 secondes à l'exception d'un graphe qui fut traité en 0.8 secondes et 4 graphes qui n'ont pas été traité endéans les cinq minutes. En ce qui concerne l'algorithme Schreier-Sims, chaque graphe orienté a été traité en moins de 3.1 secondes. Tous les graphes non orientés ont été traités en moins d'une seconde, à l'exception de deux d'entre eux, avec 4 secondes et 8 secondes.

Table 1: Comparaison sur les graphes non orientés pour les symétries de variable.

Toutes les solutions 5 min.				
	solved	unsol	total time	mean time
CSP	58%	42%	70 min.	5.95 sec.
Gen.	60,5%	39,5%	172 min.	13.95 sec.
FVS	61.8%	38.2%	101 min.	8 sec.

Table 2: Comparaison sur les graphes orientés pour les symétries de variable.

Toutes les solutions 5 min.				
	solved	unsol	total time	mean time
CSP	67%	33%	21 min.	4.31 sec.
Gen.	74%	26%	47 min.	8.87 sec.
FVS	74%	26%	40 min.	7.64 sec.

Dans nos tests, nous cherchons toutes les solutions. Une exécution est considérée comme *solved* si elle finit en moins de 5 minutes, *unsolved* sinon. Une première série d'expériences a consisté à évaluer l'impact de l'élimination des symétries de variable. Nous avons utilisé le modèle de base (CSP), dans lequel aucune symmétrie n'est éliminée, le modèle où les symétries de variable sont éliminées en postant des contraintes éliminant les générateurs (Gen.), et finalement l'élimination de toutes les symétries de variable (FSV). Les résultats de ces expériences se trouvent dans les tables 1 et 2. Dans ces exécutions, le temps de preprocessing n'a pas été considéré. La colonne 'total time' montre le temps total requis pour les instances résolues en moins de cinq minutes. La colonne 'mean time' montre le temps moyen pour les instances résolues en moins de cinq minutes.

Grâce à l'élimination des symétries de variable plus d'instances sont résolues en moins de cinq minutes, aussi bien pour les graphes orientés que pour les graphes non orientés. En ce qui concerne le temps moyen, l'élimination de toutes les symétries de variable a un avantage clair. Le fait que le temps moyen augmente est un comportement étonnant qui devrait être étudié.

L'élimination des symétries de valeur a été évalué sur l'ensemble des graphes orientés. Le tableau 4 montre que seulement un pourcent a été gagné. Ceci est peut-être dû au fait que il y a moins de symétries dans les graphes orientés que dans les graphes non orientés.

Table 3: Comparaison sur les graphes orientés pour les symétries de valeur.

Toutes les solutions 5 min.				
	solved	unsol	total time	mean time
GE-Tree	68%	32%	21 min.	4.39 sec.

Table 4: Comparaison sur des graphes non orientés pour l'élimination des symétries de variable et de valeur.

Toutes les solutions 5 min.				
	solved	unsol	total time	mean time
CSP	53,6%	46,3 %	31 min.	20.1 sec.
GE-Tree	55,3%	45,7 %	6 min.	3.21 sec.
FVS	54,9 %	45,1%	31 min.	19 sec.
GE-Tree and FVS	55,3 %	44,7%	26 min.	8.68 sec.

En ce qui concerne l'élimination des symetries de variable et de valeur ensemble, un total de 233 instances non orientées ont été traitées. Nous avons évalué l'élimination de symétries de variable et de valeur séparément, pour ensuite les considérer ensemble. Le tableau 4 montre que, comme attendu, l'élimination des symétries de valeur et des symétries de variable augmente le nombre d'instances résolues en moins de cinq minutes. Remarquons que l'élimination des symétries de valeur avec GE-Tree mène à de nouvelles instances résolues et à de meilleurs performances, avec le temps moyen qui diminue significativement. De plus, la combinaison de l'élimination des symétries de valeur et de variable ne combine pas la puissance de chacune des deux techniques prises séparément. En fait les perfomances de GE-Tree ne sont pas battues, et le temps moyen de GE-Tree est même augmenté.

De ces expériences, nous concluons que bien que l'élimination de symétries de variable et de valeur donne de meilleures performances et résout de nouvelles instances, ces techniques ne sont pas suffisantes pour générer un gain significatif d'instances résolues en moins de cinq minutes. Ce fait appelle à l'utilisation de l'élimination des symétries conditionnelles et locales.

9 Conclusion

Dans cet article, nous avons présenté des techniques pour l'élimination de symétries pour l'appariement de graphes. Des techniques de détection spécifiques ont été developpées pour les problèmes classique d'élimination des symétries de variable et de valeur. Nous montrons que les symétries de variable et de valeur peuvent être détectées en calculant l'ensemble des automorphismes sur le graphe source et le graphe cible. Nous avons aussi montré que les symétries de valeur conditionnelles peuvent être détectées en calculant l'ensemble des automorphismes sur divers sous-graphe du graphe cible. La méthode GE-Tree a été étendue pour utiliser ces symétries conditionnelles. Nous avons présenté le concept de symétrie de valeur locale, qui sont des symétries sur un sous-problème. Nous avons indiqué comment de telles nouvelles symétries peuvent être calculées et exploitées en utilisant des méthodes standard telles que GE-Tree. Nos résultats expérimentaux montrent l'amélioration obtenue pour l'élimination de symétries de variable et de valeur.

Comme perspective de recherche, nous souhaitons mener des expérimentations sur les symétries conditionnelles et sur les symétries conditionnelles de valeur, et le développement d'heuristiques pour l'intégration de ces symétries dans la recherche. Une direction de recherche intéressante est la détection automatique des symétries dans les variables à domaine représentant un graphe. Finalement, un problème ouvert est la possibilité d'éliminer les symétries dynamiques de variable.

References

[1] N. Beldiceanu, P. Flener, and X. Lorca. The tree constraint. In *Proceedings of CP-AI-OR'05*, volume LNCS 3524. Springer-Verlag, 2005.

[2] H. Cambazard and E. Bourreau. Conception d'une constrainte globale de chemin. In *10e Journ. nat. sur la résolution de problèmes NP-complets (JNPC'04)*, pages 107–121, 2004.

[3] Ronay-Dougal C.M., I.P. Gent, Kelsey T., and Linton S. Tractable symmetry breaking in using restricted search trees. *ECAI'04*, 2004.

[4] David Cohen, Peter Jeavons, Christopher Jefferson, Karen E.Petrie, and Barbara M. Smith. Symmetry definitions for constraint satisfaction problems. In van Beek [27], pages 17–31.

[5] Donatello Conte, Pasquale Foggia, Carlo Sansone, and Mario Vento. Thirty years of graph matching in pattern recognition. *IJPRAI*, 18(3):265–298, 2004.

[6] J. Crawford, M. Ginsberg, E. Luks, and A. Roy. Symmetry breaking predicates for search problem. In *Proceedings of KR'96*, 1996.

[7] Yves Deville, Grégoire Dooms, Stéphane Zampelli, and Pierre Dupont. Cp(graph+map) for approximate graph matching. *1st International Workshop on Constraint Programming Beyond Finite Integer Domains, CP2005*, 2005.

[8] Grégoire Dooms, Yves Deville, and Pierre Dupont. Cp(graph): Introducing a graph computation domain in constraint programming. *Principles and Pratice of Constraint Programming*, 2005.

[9] Ian .P. Gent, Tom Kelsey, Steve Linton, and Colva Roney-Dougal. Symmetry and consistency. In van Beek [27], pages 271–285.

[10] Ian .P. Gent, Tom Kelsey, Steve A. Linton, Iain McDonald, Ian Miguel, and Barbara M. Smith. Conditional symmetry breaking. In van Beek [27], pages 256–270.

[11] I.P. Gent. A symmetry breaking constraint for indistinguishable values. In *Proceedings of CP'01, SymCon'01 Workshop*, 2001.

[12] I.P. Gent, W. Harvey, and T. Kelsey. Groups and constraints : symmetry breaking during search. In *Proceedings of CP'02*, pages 415–430, 2002.

[13] I.P. Gent, W. Harvey, and T. Kelsey. Generic sbdd using computational group theory. In *Proceedings of CP'03*, pages 333–346, 2003.

[14] I.P. Gent and B.M. Smith. Symmetry breaking during search in constraint programming. In *Proceedings of CP'01*, pages 599–603, 2001.

[15] Javier Larrosa and Gabriel Valiente. Constraint satisfaction algorithms for graph pattern matching. *Mathematical. Structures in Comp. Sci.*, 12(4):403–422, 2002.

[16] Nikos Mamoulis and Kostas Stergiou. Constraint satisfaction in semi-structured data graphs. In Mark Wallace, editor, *CP2004*, volume 3258 of *Lecture Notes in Computer Science*, pages 393–407. Springer, 2004.

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

[18] P. Meseguer and C. Torras. Exploiting symmetries within the constraint satisfaction search. *Artificial intelligence*, 129(1-2):133–163, 2001.

[19] Jean-Francois Puget. Symmetry breaking using stabilizers. In Francesca Rossi, editor, *Proceedings of CP'03*, volume 2833 of *Lecture Notes in Computer Science*, pages 585–599. Springer, 2003.

[20] Jean-Francois Puget. Breaking all values symmetries in surjection problems. In *Proceedings of CP'05*, pages 490–504, 2005.

[21] Jean-Francois Puget. Elimination des symétries dans les problèmes injectifs. In *Proceedings des Journées Francophones de la Programmation par Contraintes*, 2005.

[22] Jean-François Puget. Automatic detection of variable and value symmetries. In van Beek [27], pages 477–489.

[23] J.-C. Regin. A filtering algorithm for constraints of difference in CSPs. In *Proc. 12th Conf. American Assoc. Artificial Intelligence*, volume 1, pages 362–367. Amer. Assoc. Artificial Intelligence, 1994.

[24] Michael Rudolf. Utilizing constraint satisfaction techniques for efficient graph pattern matching. In Hartmut Ehrig, Gregor Engels, Hans-Jörg Kreowski, and Grzegorz Rozenberg, editors, *TAGT*, volume 1764 of *Lecture Notes in Computer Science*, pages 238–251. Springer, 1998.

[25] M. Sellman. Cost-based filtering for shorter path constraints. In *Proc. of the 9th International Conference on Principles and Pratice of Constraint Programming (CP).*, volume LNCS 2833, pages 694–708. Springer-Verlag, 2003.

[26] B. Smith. Reducing symmetry in a combinatorial design problem. *Proc. CP-AI-OR'01, 3rd Int. Workshop on Integration of AI and OR Techniques in CP*, 2001.

[27] Peter van Beek, editor. *Principles and Practice of Constraint Programming - CP 2005, 11th International Conference, CP 2005, Sitges, Spain, Augustus 1-5, 2005, Proceedings*, volume 3709 of *Lecture Notes in Computer Science*. Springer, 2005.

[28] Stéphane Zampelli, Yves Deville, and Pierre Dupont. Approximate constrained subgraph matching. *Principles and Pratice of Constraint Programming*, 2005.