Programmation par contraintes pour les problèmes de plus grand sous-graphe commun

Philippe Vismara
LIRMM, UMR5506 CNRS - Université Montpellier II, Montpellier, France
MISTEA, UMR729 Montpellier SupAgro - INRA, Montpellier, France
vismara@lirmm.fr

Résumé

Les problèmes de satisfaction de contraintes ont montré leur efficacité pour résoudre des questions de comparaison de graphes comme l'isomorphisme de graphe ou de sous-graphe. La recherche du plus grand sous-graphe commun est un problème plus difficile qui a été peu abordé par des CSP. Dans cet article nous identifions différentes déclinaisons de ce problème en termes de théorie des graphes (sous-graphe induit, partiel, connexe, ...). Puis nous étudions la possibilité de les modéliser dans un solver classique en abordant notamment les questions de connexité et de symétrie. Nous présentons enfin quelques résultats expérimentaux.

Abstract

Constraint Programming has proven its efficiency to solve graph matching problems such as graph or subgraph isomorphism. It is much harder to compute maximum common subgraphs and this problem has received little attention in the CSP litterature. In this paper we discuss different variants of the problem. We consider how to model them with a conventional CSP solver and we focus on the connexity and symmetry topics. Finally, we present some experimental results.

1 Introduction

Le recours de plus en plus fréquent aux modèles de graphes pour représenter des connaissances a fait grandir le besoin d'outils efficaces pour les manipuler. La plupart des problèmes de graphes étant NP complet, il n'est pas étonnant que de nombreux travaux en programmation par contraintes se soient récemment intéressés, avec succès, à des questions comme l'isomorphisme de graphe (non classé) ou de sous-graphe. La recherche du plus grand sous-graphe commun est un problème plus difficile qui a été peu abordé par des CSP alors qu'il se pose fréquemment dans des domaines comme la Chimie [24] ou la Reconnaissance de formes [4].

Dans cet article nous identifions les différentes déclinaisons de ce problème en termes de théorie des graphes (sous-graphe induit, partiel, connexe, ...) puis nous étudions leur modélisation dans le cadre de la programmation par contraintes.

Par soucis de clarté nous nous limiterons au cas des graphes non orientés bien que cette étude soit généralisable aux graphes orientés.

2 Les problèmes de plus grand sous-graphe commun

2.1 Notations

Étant donné un graphe simple non orienté $G = (N, E)$, $N = \{1, ...n\}$ désigne l'ensemble des n sommets de G et E l'ensemble des m arêtes.

Une arête entre x et y sera généralement notée xy (sachant que $xy = yx$).

Si le graphe est étiqueté, on notera $l_G : N \cup E \to \mathcal{L}$ la fonction qui associe à chaque sommet ou arête de G un label appartenant à l'ensemble $\mathcal{L}$ de tous les labels (de tous les graphes traités).

$\forall N' \subseteq N$, on note $E(N') = \{uv \in E \mid u, v \in N'\}$

Un *sous-graphe induit* de G est un graphe $G' = (N', E')$ tel que $N' \subseteq N$ et $E' = E(N')$.

Un *graphe partiel* de G est un graphe $G' = (N, E')$ tel que $E' \subseteq E$.

Un *sous-graphe partiel* de G est un graphe $G' = (N', E')$ tel que $N' \subseteq N$ et $E' \subseteq E(N')$.

Le *graphe des arêtes* (*line graph*) de G, est le graphe $L(G) = (E, \mathcal{E})$ tel que $\mathcal{E} = \{\alpha\beta \in E \times E \mid \alpha \cap \beta \neq \varnothing\}$.

2.2 Définitions

Dans la littérature, le terme de *plus grand sous-graphe commun* peut faire référence à des notions très différentes suivant qu'il s'agit par exemple de sous-graphes *induits* ou *partiels*. Cette confusion est encore plus importante dans les articles anglophones où "sub-graph" est souvent (mais pas toujours) synonyme de sous-graphe induit.

Étant donnés deux graphes $G_A = (N_A, E_A)$ et $G_B = (N_B, E_B)$, résoudre un problème de plus grand sous-graphe commun consiste à trouver une fonction partielle injective $\mu : N_A \longrightarrow N_B$ telle que :
MCIS (induit) : $dom(\mu)$ soit de taille maximum et $\forall x, y \in dom(\mu), xy \in E_A \Leftrightarrow \mu(x)\mu(y) \in E_B$
MCCIS (induit connexe) $\equiv$ MCIS et le graphe $G' = (dom(\mu), E(dom(\mu)))$ est connexe.
MCS (partiel) : l'ensemble E' des arêtes préservées par μ soit de taille maximum avec $E' = \{xy \in dom(\mu) \times dom(\mu) \mid xy \in E_A \wedge \mu(x)\mu(y) \in E_B\}$.
MCCS (partiel connexe) $\equiv$ MCS et le graphe $G(dom(\mu), E')$ est connexe.

Si les graphes sont étiquetés, μ doit conserver les labels : $\forall x \in dom(\mu), l_{G_A}(x) = l_{G_B}(\mu(x))$ et $\forall x, y \in dom(\mu), l_{G_A}(xy) = l_{G_B}(\mu(x)\mu(y))$

Ces définitions sont illustrées par la figure 1 qui donne un exemple de solution pour chacune d'elle.

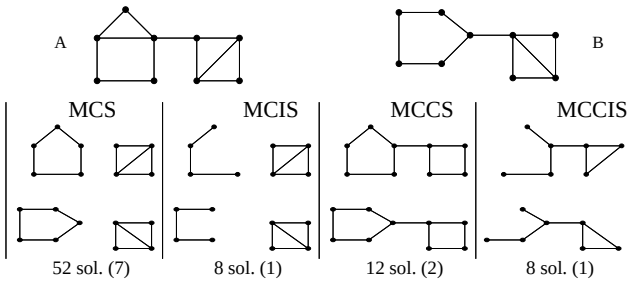


FIG. 1 – Différents plus grands sous-graphes communs

Le résultat obtenu est généralement très différent suivant la définition choisie, sans qu'il soit possible d'identifier une définition meilleure que les autres, y compris au sein d'un même champ d'application où ces notions permettent de modéliser des questions très diverses.

Par ailleurs, chaque problème admet souvent un grand nombre de solutions (52 MCS différents pour la figure 1). Or la recherche d'un plus grand sous graphe commun n'est souvent qu'une simplification d'un problème de comparaison de graphes beaucoup plus complexe. Si cette spécificité ne peut être formalisée, il devient nécessaire de confronter les résultats obtenus au regard d'un expert. Dans ce cas, il est indispensable d'éliminer les solutions symétriques. Par exemple, dans la figure 1, le nombre de solutions non symétriques est indiqué entre parenthèses.

D'un point de vue formel, nous dirons que deux solutions μ et μ' sont équivalentes à une symétrie de G_A (resp. G_B) près s'il existe un automorphisme σ_A (resp. σ_B) de G_A tel que $\mu' = \mu \circ \sigma_A$ (resp $\mu' = \sigma_B \circ \mu$).

2.3 Transformer un MCS en MCIS

La notion de *graphe des arêtes* fournit un moyen simple de transformer un problème de MCS en MCIS. La figure 2 donne un exemple d'équivalence entre les sous-graphes partiels de deux graphes et les sous-graphes induits correspondants dans leurs graphes des arêtes.

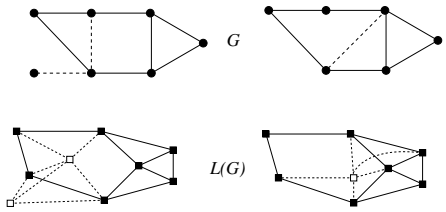


FIG. 2 – Transformer un MCS en MCIS du graphe des arêtes.

D'après le théorème de Withney (1935), deux graphes sont isomorphes si et seulement si leurs graphes des arêtes le sont, sauf si l'un des deux est K_3 et l'autre $K_{1,3}$ (interchange triangle / trinode). La figure 3 donne un exemple d'interchange trinode/triangle, si on considère uniquement les arêtes a, b et c. Par contre, les graphes A et B en entier sont bien isomorphes.

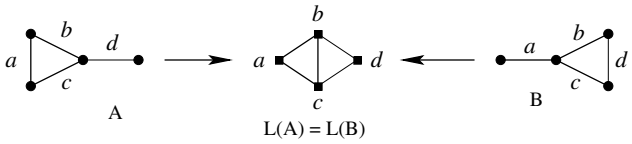


FIG. 3 – Interchange triangle / trinode (en gras)

En s'appuyant sur la preuve de ce théorème dans [11], on peut montrer que si μ est solution du MCIS entre G_A et G_B, on peut déduire de μ une unique solution du MCS entre $L(G_A)$ et $L(G_B)$ et réciproquement, pour peu que $|dom(\mu)| \geq 5$. Dans la figure 3, le morphisme $\mu(a) = a$, ..., $\mu(d) = d$ est une solution du MCIS($L(A)$, $L(B)$) mais pas une solution de MCS(A,B). Par contre, il suffit d'ajouter un sommet aux graphes A et B pour qu'un tel interchange soit impossible.

Du point de vue de la complexité, cette transformation MCS $\rightarrow$ MCIS ne simplifie pas le problème puisqu'un graphe des arêtes $L(G)$ contiendra $|E|$ sommets.

Par ailleurs il faut veiller à interdire les interchanges triangles / trinodes, surtout pour la version du problème non nécessairement connexe.

Dans le cas de graphes étiquetés, l'équivalence sera préservée si les sommets de $L(G)$ sont étiquetés en combinant le label de l'arête de G correspondante avec ceux de ces extrémités dans G ; une arête de $L(G)$ sera quant à elle étiquetée par le label du sommet correspondant dans G.

La *subdivision* est une autre façon de transformer un MCS en MCIS. Le graphe subdivisé $S(G) = (N \cup E, \mathcal{H})$ est obtenu en insérant un sommet entre les extrémités de chaque arête (cf. fig. 4). Dans ce cas, tout sous-graphe partiel de G devient un sous-graphe induit de $S(G)$ dans lequel chaque noeud issu de E possède 0 ou 2 voisins. Le graphe $S(G)$ contient donc $|E| + |N|$ sommets ce qui le rend généralement moins intéressant que le graphe des arêtes.

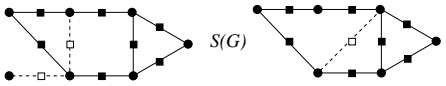


FIG. 4 – Graphe subdivisé

2.4 État de l'art

Comparé à d'autres problèmes de graphes, la recherche d'un plus grand sous-graphe a été très peu traitée dans la littérature. Dans certains domaines comme la Chimie, différentes heuristiques ont parfois été proposées sans qu'il soit possible d'en mesurer l'efficacité ou la correction. C'est pourquoi nous nous limiterons ici à présenter les deux principaux algorithmes utilisés.

La première méthode, probablement la plus répandue, consiste à transformer un MCIS en la recherche de cliques dans un graphe de compatibilité $G_{\mathcal{C}} = (N_{\mathcal{C}}, E_{\mathcal{C}})$. Ce graphe est défini sur l'ensemble des paires de sommets compatibles des deux graphes : $N_{\mathcal{C}} = \{(a, b) \in N_A \times N_B \mid l_{G_A}(a) = l_{G_B}(b)\}$. Deux sommets sont reliés dans $G_{\mathcal{C}}$ si les couples qu'ils représentent respectent la connexité des graphes G_A et G_B : $E_{\mathcal{C}} = \{(a_1, b_1)(a_2, b_2) \in N_{\mathcal{C}} \times N_{\mathcal{C}} \mid a_1a_2 \in E_A \Leftrightarrow b_1b_2 \in E_B \wedge l_{G_A}(a_1a_2) = l_{G_B}(b_1b_2)\}$.

Toute clique de $G_{\mathcal{C}}$ correspond naturellement à un morphisme qui respecte la définition du MCIS. Le problème du MCIS se ramène alors à rechercher des cliques de taille maximum dans $G_{\mathcal{C}}$ [18]. Cette approche est par exemple utilisée dans [10]. Il existe de nombreux travaux sur la détermination des cliques maximum [1] y compris dans le cadre de la programmation par contraintes [26]. Pour résoudre le problème du MCCIS, Koch [14] propose une variante de l'algorithme de Bron et kerbosch [2] afin d'obtenir des cliques de $G_{\mathcal{C}}$ qui correspondent à un sous-graphe connexe dans G_A ou G_B.

Le second type de méthodes regroupe toutes les approches qui ne construisent pas de graphe de compatibilité mais explorent directement l'espace de recherche avec un mécanisme de "backtrack". L'algorithme de Mc Gregor [19], initialement conçu pour résoudre un MCS, est un des plus cité pour résoudre un MCIS. Cet algorithme utilise une démarche analogue à la programmation par contraintes qui s'apparente au *forward checking*. Plus récemment, un autre algorithme de backtrack a été proposé par [15] avec là aussi une approche très similaire au *forward checking* et une optimisation de l'exploration qui revient à choisir, à chaque étape, la variable dont le domaine est de taille minimum.

Les deux types d'approches que nous venons d'évoquer ont été comparées dans [5] sur une base de 81400 paires de graphes aléatoires ayant différentes caractéristiques (densité, régularités, ...). Leur conclusion est qu'aucun des algorithmes étudiés ne se distingue clairement, le classement s'inversant suivant le type de graphe. On peut également remarquer que bon nombre d'algorithmes d'énumération de cliques ont une approche de type "backtrack". Tant que le graphe de compatibilité n'est pas trop grand, les deux types d'algorithmes peuvent donc être amenés à explorer des espaces de recherche très semblables.

Par ailleurs, nous avons vu que plusieurs algorithmes dédiés au MCIS emploient des techniques très analogues à celles utilisées en programmation par contraintes. Ce paradigme n'est pourtant jamais évoqué dans la littérature sur les problèmes de plus grand sous-graphe commun. La prochaine section s'intéresse donc à la résolution de ces problèmes par un CSP.

3 Un CSP pour déterminer le plus grand sous-graphe commun

Ces dernières années, plusieurs problèmes d'appariement de graphes ont fait l'objet d'une modélisation par un réseau de contraintes. Qu'il s'agisse d'isomorphisme de sous-graphe [25, 28, 16, 37, 29] ou d'isomorphisme de graphe [30], ces travaux reposent sur un modèle dans lequel les variables correspondent à des sommets des graphes traités. Une approche plus originale a été proposée dans [9] pour modéliser des problèmes de sous-graphe dans CP(Graph) [8] puis d'appariement de graphes dans CP(Map)[35]. Dans ce modèle, les variables sont des graphes ou des fonctions d'appariement entre ces graphes. Cette approche a montré son efficacité pour résoudre des problèmes comme l'isomorphisme de sous-graphe [7] mais n'a pas été généralisée aux problèmes de sous-graphe commun.

Plus récemment, un formalisme général a été proposé dans [17] pour modéliser les principaux problèmes d'appariement de graphes (graph matching) incluant les problèmes cités précédemment ainsi que le MCS et le MCIS. Cet élégant modèle considère un *matching* comme un sous-ensemble de $N_A \times N_B$ sur lequel il est possible de définir une série de contraintes de base. Les problèmes d'appariement peuvent alors être construits en combinant ces différentes contraintes. Ce modèle a été mis en oeuvre dans Comet [31], un langage de programmation qui combine contraintes et recherche locale. Les résultats ont montré l'efficacité de cette approche pour résoudre l'isomorphisme de sous-graphe. Quant au problème du MCS, il a été implémenté par un algorithme tabou permettant d'obtenir une solution approchée en temps raisonnable (quelques dizaines de secondes) pour des graphes de taille moyenne (50 sommets) extraits de la base de tests de [5].

La mise au point d'approches efficaces pour déterminer des plus grands sous-graphes communs semble donc toujours d'actualité. En attendant l'émergence d'outils dédiés à ces problèmes, il est intéressant de voir dans quelle mesure il est déjà possible d'en résoudre certains, dans le cadre classique de la programmation par contraintes. La suite de cette section a pour but d'identifier les principales questions à résoudre et de proposer une modélisation de ces problèmes. L'objectif n'est pas de trouver la meilleure modélisation mais d'en proposer une suffisamment simple pour être rapidement mise en oeuvre.

3.1 Sous-graphes induits (MCIS)

Pour résoudre un MCIS, entre $G_A = (N_A, E_A)$ et $G_B = (N_B, E_B)$, on peut construire un CSP dont les variables correspondent aux sommets de G_A ou de G_B ou même des deux. Nous allons utiliser cette seconde solution qui facilite la modélisation du problème, notamment pour la recherche de solutions connexes. Par ailleurs elle évite d'introduire une distinction arbitraire entre les deux graphes.

L'ensemble des variables du CSP se décompose donc en deux sous-ensembles : $X_A = \{a_i\}_{i \in N_A}$ pour N_A et $X_B = \{b_j\}_{j \in N_B}$ pour N_B.

On notera $n_A = |N_A|$ (resp $n_b = |N_B|$) et on supposera que $N_A = 1..n_a$.

Pour obtenir une fonction partielle et injective, on associe une valeur spéciale, propre à chaque variable, afin d'identifier les sommets perdus :

$$\forall i \in 1..n_A,\ D(a_i) = N_B \cup \{n_B + i\}$$

Si les graphes G_A et G_B sont étiquetés, on ne garde que les valeurs compatibles :

$$D(a_i) = \{k \in N_B \mid l_{G_B}(k) = l_{G_A}(i)\} \cup \{n_B + i\}$$

Pour résoudre le MCIS, on pose les contraintes suivantes :

channeling on définit une contrainte de channeling partiel entre les variables de X_A et X_B, $\forall a_i \in X_A, b_j \in X_B, (a_i \leq n \wedge a_i = j) \Leftrightarrow (b_j \leq n \wedge b_j = i)$

différence puisque chaque variable de X_A ou de X_B possède sa propre valeur spéciale, on obtiendra une solution injective en définissant une contrainte de différence $allDifferent(a_1, ..., a_{n_A})$ et $allDifferent(b_1, ..., b_{n_B})$

voisinage l'isomorphisme des sous-graphes induits est garanti par $n_A + n_B$ contraintes binaires :
- $\forall ik \in E_A, (a_i \leq n_B \wedge a_k \leq n_B) \Rightarrow a_i a_k \in E_B$
- $\forall jz \in E_B, (b_j \leq n_A \wedge b_z \leq n_A) \Rightarrow b_j b_z \in E_A$

Si les graphes sont étiquetés, ces contraintes deviennent :
- $\forall ik \in E_A, (a_i \leq n \wedge a_k \leq n)$
  $\Rightarrow (a_i a_k \in E_B \wedge l_{G_A}(ik) = l_{G_B}(a_i a_k))$
- $\forall jz \in E_B, (b_j \leq n \wedge b_z \leq n)$
  $\Rightarrow (b_j b_z \in E_A \wedge l_{G_B}(jz) = l_{G_A}(b_j b_z))$

Pour obtenir une solution de taille maximale, on définit une variable *taille* qui compte le nombre de sommets conservés dans G_A et G_B (donc $D(taille) = 1..(n_A + n_B)$). Une contrainte *among* permet de garantir que *taille* est le nombre de variables dont la valeur est inférieure à n_A ou n_B.

3.2 Sous-graphes induits connexes (MCCIS)

On trouve dans la littérature peu de méthodes pour définir une contrainte de connexité.

Dans CP(Graph)[8] deux solutions ont été étudiées pour la recherche de sous-graphes d'un graphe donné (sans morphisme). La première repose sur la fermeture transitive de la relation de connexité. Elle suppose de définir $O(n^3)$ contraintes disjonctives N-aires et $O(n^4)$ variables booléennes, ce qui rend cette solution inadaptée aux grands graphes. La seconde est une contrainte globale qui réalise le filtrage par un simple parcours en profondeur du graphe ($O(m + n)$ en temps par filtrage). Dès qu'au moins un sommet est conservé dans le sous-graphe, on peut éliminer tous les sommets qui ne sont pas dans la même composante connexe. On peut aussi conserver tous les points d'articulation[1] (et les isthmes[2]) situés entre deux sommets conservés dans la solution courante.

L'algorithme utilisé pour ce filtrage n'est pas détaillé mais il reste facile à concevoir à partir d'un parcours en profondeur (Tarjan), avec calcul d'un ordre de *pre* et *post* d'exploration des sommets. En partant d'un sommet d tel que $a_d \leq n_B$, un point d'articulation sera

[1] point d'articulation = sommet dont la suppression déconnecte le graphe

[2] isthme = arête dont la suppression déconnecte le graphe

conservé si le descendant qui a permis de le détecter faisait partie d'une composante 2-connexe contenant elle aussi un sommet s tel que $a_s \leq n_B$.

Une contrainte globale similaire est proposée dans [20] pour le calcul d'un graphe partiel (tous les sommets sont conservés). Un parcours en profondeur est utilisé pour déterminer un arbre recouvrant du graphe. Cet arbre est stocké dans une structure de données réversible avec une série de compteurs indiquant le nombre de cycles fondamentaux[3] passant par chaque arête. Cette structure (en $O(n^2)$) permet de détecter la déconnexion du graphe partiel sans lancer un parcours à chaque filtrage (sauf en cas de suppression d'une arête de l'arbre).

Une autre façon de garantir la connexité de la solution [32], mais sans filtrage supplémentaire, est d'utiliser un ordre d'instanciation des variables qui choisisse la nouvelle variable à instancier parmi les voisines des variables déjà instanciées (avec une valeur $a_s \leq n_B$). Une contrainte globale permet de maintenir à jour la composante connexe en cours de construction ainsi que l'ensemble des variables voisines non instanciées (nécessite des structures de données réversibles de taille $O(n)$). Lorsque ce voisinage devient vide, il suffit de vérifier que toutes les variables instanciées font partie de la composante connexe.

3.3 Éliminer les symétries

Dès qu'il s'agit d'énumérer toutes les solutions, il est indispensable d'éliminer celles qui sont équivalentes à un automorphisme de G_A ou de G_B près. Cette élimination peut être réalisée a posteriori mais avec un coût important. Par ailleurs, éliminer ces symétries au cours de la résolution peut fortement limiter l'espace de recherche dans la mesure où les symétries du CSP sont généralement équivalentes à celles des graphes traitées. Il est cependant possible de supprimer les solutions symétriques d'un CSP sans nécessairement éliminer toutes les branches symétriques de l'arbre de recherche.

Une des démarches les plus courantes pour éliminer les symétries de variables consiste à poser des contraintes lexicographiques [6]. Étant donné un ordre $x_{i_1}, \ldots x_{i_n}$ sur les variables, on pose, pour chaque symétrie de variable σ, une contrainte de la forme $x_{i_1}, \ldots x_{i_n} \leq_{lex} x_{\sigma(i_1)}, \ldots x_{\sigma(i_n)}$. Malheureusement, il peut exister un nombre exponentiel de symétries.

Cette même méthode peut être utilisée pour éliminer toute symétrie de valeur θ en posant une contrainte de la forme $x_{i_1}, \ldots x_{i_n} \leq_{lex} \theta(x_{i_1}), \ldots \theta(x_{i_n})$ [33]. Mais maintenir la GAC est dans ce cas NP-Complet [34]. Quant aux combinaisons de symétries entre symétrie de variables σ et symétrie de valeur θ, on peut les éliminer en posant des contraintes de la forme $x_{i_1}, \ldots x_{i_n} \leq_{lex} \theta(x_{\sigma(i_1)}), \ldots \theta(x_{\sigma(i_n)})$ [23].

Pour la recherche d'un isomorphisme de sous-graphe de G_p dans G_t, les auteurs de [36] ont montré qu'il est possible d'éliminer les symétries du graphe G_p, sur lequel sont définies les variables x_i, en ajoutant un nombre linéaire de contraintes. Cette méthode utilise les résultats de [22] sur l'élimination des symétries de variables dans les problèmes injectifs. Elle nécessite le calcul du groupe $\mathcal{A}_{G_p}$ des automorphismes de G_p (par exemple avec le logiciel NAUTY) et l'algorithme de Schreier-Sims pour construire une chaîne de stabiliseurs. Pour chaque sommet i de G_P tel que l'ensemble $\mathcal{S}_i = \{j \neq i \mid \exists \sigma \in \mathcal{A}_{G_p} \text{ tq } i = \sigma(j) \wedge \forall k < j, \sigma(k) = k\}$ n'est pas vide, on définit $r(i) = \max(j \in \mathcal{S}_i)$. On peut noter que $r(i) < i$. Pour éliminer toutes les symétries de $\mathcal{A}_{G_p}$, il suffit de poser une contrainte de la forme $x_{r(i)} < x_i$ pour chaque $r(i)$ ainsi défini.

Cette méthode peut être utilisée pour éliminer les symétries du CSP permettant de résoudre un MCIS. Si on considère, l'ensemble X_A des variables liées à G_A et les contraintes binaires définies pour chaque arête de E_A, on obtient un sous-problème de contraintes dont les symétries sont équivalentes à celles de G_A. On peut alors éliminer les symétries de ce CSP par des contraintes de la forme $a_{r_A(i)} < a_i$, avec $r_A(i) < i$ obtenu comme précédemment à partir du groupe d'automorphismes $\mathcal{A}_{G_A}$. Ces contraintes sont compatibles avec les valeurs spéciales de la forme $i + n_B$ ajoutées aux domaines $D(a_i)$.

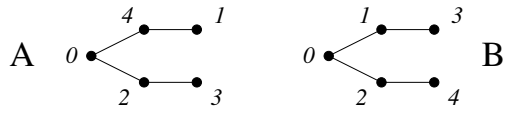
Le même raisonnement peut être employé pour éliminer les symétries de G_B en considérant le sous-problème correspondant aux variables de X_B. Les contraintes seront de la forme $b_{r_B(j)} < b_j$.

Malheureusement, il n'est pas possible d'utiliser cette méthode en même temps sur G_A et sur G_B. Par exemple, dans la figure 5, la méthode proposée par Puget [22] permet d'éliminer les symétries de G_A en posant l'unique contrainte $a_1 < a_3$. De même, les symétries de G_B sont éliminées par la contrainte $b_1 < b_2$. Or le CSP n'admet que deux solutions qui chacune respecte ou viole une des deux contraintes :

- $a_0 = 0,\ a_1 = 3,\ a_2 = 2,\ a_3 = 4,\ a_4 = 1$
  $b_0 = 0,\ b_1 = 4,\ b_2 = 2,\ b_3 = 1,\ b_4 = 3$
- $a_0 = 0,\ a_1 = 4,\ a_2 = 1,\ a_3 = 3,\ a_4 = 2$
  $b_0 = 0,\ b_1 = 2,\ b_2 = 4,\ b_3 = 3,\ b_4 = 1$

On peut remarquer que l'élimination des symétries de variables du CSP réduit à X_B est équivalente à l'élimination ses symétries de valeurs du CSP réduit à X_A. Nous sommes donc confrontés au problème de l'élimination conjointe des symétries de variables et de

[3] étant donné un arbre recouvrant $T = (N, E')$ de $G = (N, E)$ toute arête de $E \setminus E'$ ajoutée à T forme un seul cycle dit *fondamental*



$\mathcal{A}_{G_A} = \left\{ \begin{array}{ll} \text{id} & 01234 \\ \sigma & 03412 \end{array} \right.$ $\mathcal{A}_{G_B} = \left\{ \begin{array}{ll} \text{id} & 01234 \\ \sigma' & 02143 \end{array} \right.$

$r_A(3) = 1$ $r_B(2) = 1$

FIG. 5 – Un contre-exemple

valeurs d'un CSP.

Nous avons vu que la méthode proposée par Puget dans [22] permet d'éliminer les symétries de variables liées par un AllDiff, en posant un nombre linéaire de contraintes. L'auteur a montré que cette approche est compatible avec la méthode GE-Tree décrite dans [27] pour éliminer les symétries de valeurs. Cette méthode nécessite le calcul, à chaque noeud de l'arbre de recherche, des symétries de valeurs invariantes ($\theta(v) = v$) pour les valeurs déjà affectées. Ces symétries induisent des classes d'équivalence sur les valeurs restant dans le domaine des variables non instanciées. L'instanciation de la prochaine variable est alors limitée à un seul représentant de chacune de ces classes, éliminant ainsi les branches symétriques.

Dans [21], Puget propose d'éliminer les symétries de valeurs dans les problèmes surjectifs. Il définit un ensemble $\{z_j\}$ de variables supplémentaires, en plus des variables $\{x_i\}$ du CSP, et une contrainte de channeling qui s'écrit : $(x_i = j) \leftrightarrow (z_j = i)$ dans le cas d'un AllDiff sur les $\{x_i\}$. Les symétries de valeurs du CSP étant équivalentes aux symétries des variables z_j, on peut les supprimer en posant un nombre linéaire de contraintes comme dans [22].

L'analogie avec les variables a_i et b_i du CSP MCIS décrit au paragraphe 3.1 est évidente et le contre-exemple de la figure 5 s'applique aussi. Il a d'ailleurs été démontré dans [13], que l'utilisation conjointe de contraintes lexicographiques pour éliminer les symétries de variables avec la méthode de Puget [21] pour supprimer les symétries de valeurs peut faire perdre des solutions non symétriques. Cette incompatibilité interdit d'éliminer les symétries de variables et de valeurs du MCIS en utilisant, dans les deux cas, la méthode de Puget pour ne poser qu'un nombre linéaire de contraintes.

Au final, il n'est donc pas toujours possible d'éliminer toutes les symétries de façon efficace. Le choix de la méthode employée dépendra fortement de la taille des groupes de symétrie de chaque graphe.

3.4 Sous-graphes partiels (MCS)

Une première méthode pour résoudre un MCS consiste à le considérer comme un MCIS dans lequel les contraintes binaires d'adjacence peuvent être violées [17].

Si on reste dans le cadre d'un CSP plus classique, on peut résoudre le MCS en se ramenant à un MCIS sur les graphes des arêtes, grâce à la transformation présentée au paragraphe 2.3.

Pour trouver un MCS connexe (MCCS) entre G_A et G_B il suffit de chercher un MCCIS entre $L(G_A)$ et $L(G_B)$. En dehors du cas trivial de l'interchange de la figure 3, toute solution contenant au moins 5 sommets n'admet pas d'interchange.

Dans le cas d'un MCS quelconque (non-connexe), le nombre d'interchanges possibles est beaucoup plus important et une vérification a posteriori ne garantit pas une solution maximale. On doit donc ajouter des contraintes pour interdire les interchanges. Sans le channeling, il faudrait définir une contrainte pour chaque triplet d'arêtes afin de garantir qu'un trinode ou triangle sera préservé par la fonction d'appariement. Grâce au channeling, il suffit de créer une contrainte ternaire par triangle de chacun des deux graphes G_A et G_B, l'énumération initiale des cycles de taille 3 étant réalisée en temps linéaire [3].

4 Résultats expérimentaux et discussion

Le modèle de MCIS présenté dans cet article a été implémenté en Java en utilisant le solveur CHOCO [12] qui fournit la plupart des contraintes nécessaires. Seule la contrainte de *partial channeling* a du être adaptée à partir de la contrainte *Channeling* présente dans CHOCO.

La contrainte de connexité à été implémentée par une contrainte globale réalisant le parcours en profondeur évoqué au paragraphe 3.2.

L'élimination des symétries a été réalisée en combinant la librairie NAUTY et une implementation non optimisée de l'algorithme de Schreier-Sims.

Quant au MCS, il a été programmé en appliquant le MCIS sur le graphe des arêtes et en y ajoutant les contraintes ternaires interdisant un interchange.

Cette étape d'implémentation a montré qu'il est tout a fait possible de programmer un problème de plus grand sous-graphe commun avec un solveur comme CHOCO. La partie la plus complexe n'est d'ailleurs pas liée aux CSP mais à la mise au point des algorithmes de détection des symétries.

Pour évaluer l'efficacité de ces modèles, nous avons utilisé la base de graphes aléatoires construite par [5] pour le problème du MCIS. Cette base contient des séries homogènes de 100 couples de graphes entièrement étiquetés (une étiquette différente par sommets), de 10 à 100 sommets. Sur ces graphes, tous les problèmes modélisés ont été résolus en moins d'une secondes avec

un processeur à 3 Ghz. Pour réduire le nombre d'étiquettes (par exemple à la moitie du nombre de sommets), les auteurs de la base de données proposent de remplacer chaque étiquette (entre 0 et 2^{16}) par sa valeur modulo le nombre de valeurs souhaitées. Mais cette réduction ne garantit pas d'obtenir le nombre de labels voulus, notamment au niveau des arêtes et il est difficile d'exploiter les résultats. Nous avons donc choisi de comparer les graphes en supprimant toutes les étiquettes.

D'une manière globale, il existe une très grande disparité de temps de calcul, même au sein d'une même classe de graphes. Les résultats préliminaires qui suivent sont donc à interpréter avec prudence.

Les tableaux ci-dessous concernent la classe "mcs90_r005" de graphes peu denses de 20, 25 et 30 sommets et dont les couples ont 90% de parties communes. Étant peu denses, ces graphes ont un nombre d'arêtes proches du nombre de sommets ce qui permet d'avoir des instances de tailles semblables pour les versions "induit" et "partiel" des problèmes.

	MCCIS		
n	*GAC*	*BC*	*Clique*
20	$0,9$	$0,5$	$4,3$
25	$9,6$	$4,5$	$26,1$
30	$134,5$	$61,8$	> 1000

Pour la contrainte AllDiff, nous avons testé la différence entre les versions GAC et BC, cette dernière semblant plus rapide.

Dans le cas du MCCIS, la méthode reposant sur la recherche de cliques dans le graphe de compatibilité a été implémentée, en utilisant la variante de l'algorithme de Bron et Kerbosh proposée par [14]. Les résultats sont moins bons que ceux obtenus par les CSP. Il serait interessant de vérifier si des algorithmes récents de recherche de cliques ne permettraient pas de réduire cet écart.

Le tableau suivant montre que les temps de calcul obtenus pour le MCCIS ou le MCCS sont meilleurs que ceux des versions non connexes, ce qui est cohérent avec la combinatoire inhérente à ces problèmes. On peut également en conclure que la méthode de filtrage utilisée pour la contrainte de connexité, bien que perfectible, est déjà utilisable en l'état.

n	m	*MCIS*		*MCCS*		*MCS*	
		GAC	*BC*	*GAC*	*BC*	*GAC*	*BC*
20	21	$6,7$	$4,3$	$2,69$	$1,7$	$12,9$	$8,7$
25	29	$67,1$	$45,7$	$120,0$	$104,9$	375	348

L'élimination des symétries a été testée en appliquant la méthode de [22] sur un seul des graphes comparés (celui ayant le plus grand nombre d'automorphismes). Malheureusement, les graphes testés possèdent pas ou peu de symétries et si les temps de calculs sont meilleurs, ils ne sont pas significatifs.

Au final, on peut s'interroger sur la pertinence de la base de données de tests, notamment du fait de l'impossibilité de tenir compte des étiquettes qui sont souvent présentes dans des problèmes du monde réel. La grande variabilité des résultats obtenus sur des graphes théoriquement semblables pose également problème.

5 Conclusion

Dans cet article nous avons proposé une modélisation du problème de plus grand sous graphe commun dans le cadre classique de la programmation par contraintes. Les premières expérimentations sur des graphes aléatoires ont montré l'efficacité de cette démarche sur des instances assez simples mais mériteraient d'être poursuivies pour en mesurer les limites.

Cela étant, il serait probablement plus intéressant d'expérimenter ce modèle sur des problèmes réels pour lesquels des contraintes complémentaires pourraient être identifiées et formalisées. Il serait alors possible de disposer d'instances de problèmes réels pour mieux évaluer les différentes approches actuellement développées par la communauté de la programmation par contraintes pour résoudre les MCIS et autres MCS.

Références

[1] I. M. Bomze, M. Budinich, P. M. Pardalos, and M. Pelillo. The maximum clique problem. In D.-Z. Du and P. M. Pardalos, editors, *Handbook of Combinatorial Optimization (Supplement Volume A)*, pages 1–74. Kluwer Academic, 1999.

[2] C. Bron and J. Kerbosch. Finding all cliques of an undirected graph. *Communication of the ACM*, 16(9) :575–579, 1973.

[3] N. Chiba, T. Nishizeki, S. Abe, and T. Ozawa. A linear algorithm for embedding planar graphs using PQ-trees. *Journal of computer and system sciences*, 30 :54–76, 1985.

[4] D. Conte, P. Foggia, C. Sansone, and M. Vento. Thirty years of graph matching in pattern recognition. *IJPRAI*, 18(3) :265–298, 2004.

[5] D. Conte, P. Foggia, and M. Vento. Challenging complexity of maximum common subgraph detection algorithms : A performance analysis of three algorithms on a wide database of graphs. *Journal of Graph Algorithms and Applications*, 11(1) :99–143, 2007.

[6] J. Crawford, M. Ginsberg, E. Luks, and A. Roy. Symmetry-breaking predicates for search problems. In *KR*, pages 148–159. Morgan Kaufmann, 1996.

[7] Y. Deville, G. Dooms, and S. Zampelli. Combining two structured domains for modeling various

graph matching problems. In *Proceedings CSCLP 2007*, volume 5129 of *Lecture Notes in Computer Science*, pages 76–90, 2008.

[8] G. Dooms. *The CP(Graph) Computation Domain in Constraint Programming.* PhD thesis, Université catholique de Louvain, 2006.

[9] G. Dooms, Y. Deville, and P. Dupont. Cp(graph) : Introducing a graph computation domain in constraint programming. In *Proc CP 2005*. Springer Verlag, 2005.

[10] Paul J. Durand, Rohit Pasari, Johnnie W. Baker, and Chun che Tsai. An efficient algorithm for similarity analysis of molecules. *Internet Journal of Chemistry*, 2(17), June 1999.

[11] F. Harary. *Graph Theory.* Addison-Wesley, 1969.

[12] Narendra Jussien, Guillaume Rochart, and Xavier Lorca. Choco : an Open Source Java Constraint Programming Library. In *CPAIOR'08 Workshop on Open-Source Software for Integer and Contraint Programming (OSSICP'08) CPAIOR'08 Workshop on Open-Source Software for Integer and Contraint Programming (OSSICP'08)*, pages 1–10, Paris, France France, 2008.

[13] G. Katsirelos, N. Narodytska, and T. Walsh. On the complexity and completeness of static constraints for breaking row and column symmetry. In *Proceedings of the 16th international conference on Principles and practice of constraint programming*, CP'10, pages 305–320. Springer-Verlag, 2010.

[14] I. Koch. Enumerating all connected maximal common subgraphs in two graphs. *Theoretical Computer Science*, 250 :1–30, 2001.

[15] Evgeny B. Krissinel and Kim Henrick. Common subgraph isomorphism detection by backtracking search. *Software : Practice and Experience*, 34(6) :591–607, 2004.

[16] J. Larossa and G. Valiente. Constraint satisfaction algorithms for graph pattern matching. *Math. Struct. Comput. Sci.*, 12(4) :403–422, 2002.

[17] V. le Clément, Y. Deville, and C. Solnon. Constraint-based graph matching. In *Proc. Principles and Practice of Constraint Programming - CP 2009*, volume 5732 of *Lecture Notes in Computer Science*, pages 274–288, 2009.

[18] G. Levi. A note on the derivation of maximal common subgraphs of two directed or undirected graphs. *Calcolo*, 9(4) :341–352, 1972.

[19] James J. McGregor. Backtrack search algorithm and the maximal common subgraph problem. *Software Practice and Experience*, 12 :23—34, 1981.

[20] P. Prosser and C. Unsworth. A connectivity constraint using bridges. *Frontiers in Artificial Intelligence and Applications*, 141 :707, 2006.

[21] J.-F. Puget. Breaking all value symmetries in surjection problems. In *Proceedings of the 11th international conference on Principles and practice of constraint programming*, CP'05, pages 490–504, 2005.

[22] J.-F. Puget. Breaking symmetries in all different problems. In *IJCAI*, pages 272–277, 2005.

[23] J.-F. Puget. An efficient way of breaking value symmetries. In *AAAI*, 2006.

[24] J. W. Raymond and P. Willett. Maximum common subgraph isomorphism algorithms for the matching of chemical structures. *Journal of Computer-Aided Molecular Design*, 16(7) :521–533, 2002.

[25] J.-C. Régin. *Développement d'outils algorithmiques pour l'Intelligence Artificielle et application Ã la chimie organique.* Thèse de doctorat, Université Montpellier II, 1995.

[26] J-C. Régin. Using constraint programming to solve the maximum clique problem. In *Principles and Practice of Constraint Programming - CP 2003, LNCS 2833*, pages 634–648. Springer, 2003.

[27] C. M. Roney-Dougal, I. P. Gent, T. Kelsey, and S. Linton. Tractable symmetry breaking using restricted search trees. In R. López de Mántaras and L. Saitta, editors, *ECAI*, pages 211–215. IOS Press, 2004.

[28] M. Rudolf. Utilizing constraint satisfaction techniques for efficient graph pattern matching. In *6th International Workshop on Theory and Application of Graph Transformations*, volume 1764 of *Lecture Notes in Computer Science*, pages 381–394. Springer-Verlag, 1998.

[29] C. Solnon. Alldifferent-based filtering for subgraph isomorphism. *Artificial Intelligence*, 174(12-13) :850 – 864, 2010.

[30] Sébastien Sorlin and Christine Solnon. A parametric filtering algorithm for the graph isomorphism problem. *Constraints*, 13 :518–537, 2008.

[31] Pascal Van Hentenryck and Laurent Michel. *Constraint-Based Local Search.* The MIT Press, 2005.

[32] Philippe Vismara and Benoît Valery. Finding maximum common connected subgraphs using clique detection or constraint satisfaction algorithms. In *Modelling, Computation and Optimization in Information Systems and Management Sciences, MCO 2008 Proceedings*, volume 14 of

Communications in Computer and Information Science, pages 358–368. Springer, 2008.

[33] Toby Walsh. General symmetry breaking constraints. In *CP*, pages 650–664, 2006.

[34] Toby Walsh. Breaking value symmetry. In *Proceedings of the 13th international conference on Principles and practice of constraint programming*, CP'07, pages 880–887. Springer-Verlag, 2007.

[35] S. Zampelli. *A Constraint Programming Approach to Subgraph Isomorphism.* PhD thesis, Université catholique de Louvain, juin 2008.

[36] S. Zampelli, Y. Deville, M. R. Saïdi, and B. Benhamou. Symmetry breaking in subgraph isomorphism. In *SymCon'07, the Seventh International Workshop on Symmetry and Constraint Satisfaction Problem. A Satellite Workshop of CP 2007.*, 2007.

[37] S. Zampelli, Y. Deville, and C. Solnon. Solving subgraph isomorphism problems with constraint programming. *Journal of Constraints*, 2010.