Modélisation et résolution de problèmes de plus grands sous-graphes communs à l'aide de la programmation par contraintes

Samba Ndojh Ndiaye et Christine Solnon
Université de Lyon, CNRS
Université Lyon 1, LIRIS, UMR5205, F-69622, France
samba-ndojh.ndiaye@liris.cnrs.fr;christine.solnon@liris.cnrs.fr

Résumé

La distance entre deux graphes est généralement définie par rapport à la taille d'un plus grand sous-graphe commun. Ce sous-graphe commun peut être un sous-graphe induit, obtenu en supprimant des sommets, ou un sous-graphe partiel, obtenu en supprimant des arcs et des sommets. Dans cet article, nous modélisons ces deux problèmes dans un cadre unifié à l'aide de la contrainte souple *allDiff*. Nous introduisons également différents niveaux de propagation, et nous les comparons expérimentalement sur différents types de graphes.

Abstract

The distance between two graphs is usually defined by means of the size of a largest common subgraph. This common subgraph may be an induced subgraph, obtained by removing nodes, or a partial subgraph, obtained by removing arcs and nodes. In this paper, we introduce two soft CSPs which model these two maximum common subgraph problems in a unified framework. We also introduce and compare different CP models, corresponding to different levels of constraint propagation.

1 Introduction

Les graphes sont utilisés dans de nombreuses applications pour représenter des objets structurés tels que, par exemple, des molécules, des images ou des réseaux d'interactions biochimiques. Dans beaucoup de ces applications, il est nécessaire de disposer d'une mesure de distance entre deux graphes, distance qui est souvent définie par rapport à la taille du plus grand sous-graphe commun aux deux graphes. Plus précisément, on peut soit rechercher le plus grand sous-graphe induit (ayant le plus grand nombre de sommets), soit rechercher le plus grand sous-graphe partiel (ayant le plus grand nombre d'arcs). Ces deux problèmes sont NP-difficiles dans le cas général [7], et ont fait l'objet de nombreux travaux, notamment en bioinformatique et chemoinformatique [15, 13].

L'utilisation de la programmation par contraintes pour la résolution du problème d'isomorphisme de sous-graphes a été récemment étudiée dans [18, 16]. Ces travaux ont notamment montré l'intérêt de la contrainte globale *allDiff* pour résoudre ce genre de problème. De fait, cette contrainte a été initialement introduite par Régin pour résoudre ce problème [15].

Dans cet article, nous étudions comment utiliser la programmation par contraintes pour résoudre des problèmes de recherche de plus grands sous-graphes communs. Nous nous intéressons tout particulièrement à l'apport de la contrainte globale *allDiff* pour la résolution de ces problèmes. Dans la section 2, nous rappelons quelques définitions et nous décrivons les principales approches existantes. Dans la section 3, nous montrons comment modéliser ces deux problèmes en termes de contraintes dures et souples. Cette modélisation est faite dans un cadre unifié pour les deux problèmes. Dans la section 4, nous introduisons différents niveaux de propagation de contraintes. Dans la section 5, nous comparons expérimentalement ces différents modèles.

2 Contexte

2.1 Définitions

Un graphe $G = (N, A)$ se compose d'un ensemble de nœuds N et un ensemble d'arcs $A \subseteq N \times N$. Nous considérons implicitement des graphes orientés, dont

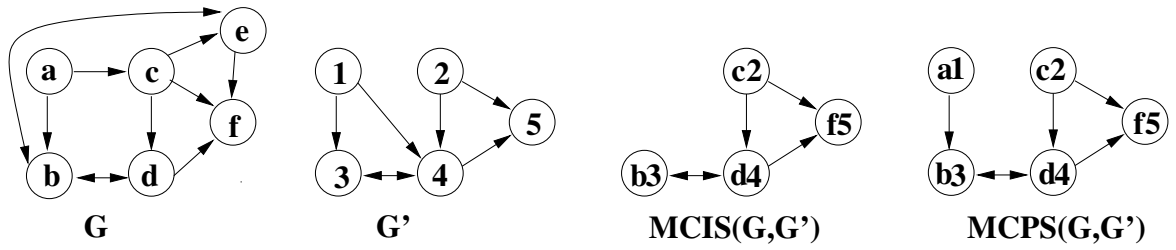

FIGURE 1 – Exemple de MCIS et MCPS pour deux graphes G et G'.

les arcs sont des couples orientés de nœuds. Les résultats présentés dans la suite peuvent être aisément généralisés aux graphes non orientés en associant à chaque arête $\{u, v\}$ deux arcs (u, v) et (v, u).

Soient deux graphes $G = (N, A)$ et $G' = (N', A')$. G est *isomorphe* à G' s'il existe une fonction bijective $f : N \to N'$ qui préserve les arcs, *i.e.*,

$$\forall (u, v) \in N \times N, (u, v) \in A \Leftrightarrow (f(u), f(v)) \in A'$$

Un *sous-graphe induit* est obtenu en supprimant des nœuds, *i.e.*, G' est un sous-graphe induit de G si $N' \subseteq N$ et $A' = A \cap N' \times N'$. Un *sous-graphe partiel* est obtenu en supprimant des nœuds et des arcs, *i.e.*, G' est un sous-graphe partiel de G si $N' \subseteq N$ et $A' \subseteq A \cap N' \times N'$.

Nous notons $G_{\downarrow S}$ le sous-graphe obtenu en gardant un sous-ensemble S de composants de G : si S est un sous-ensemble de nœuds (*i.e.*, $S \subseteq N$), alors $G_{\downarrow S}$ est le sous-graphe induit obtenu en gardant ces nœuds (*i.e.*, $G_{\downarrow S} = (S, A \cap S \times S)$) ; si S est un sous-ensemble d'arcs (*i.e.*, $S \subseteq A$), alors $G_{\downarrow S}$ est le sous-graphe partiel obtenu en gardant ces arcs (*i.e.*, $G_{\downarrow S} = (N_S, S)$ avec $N_S = \{u \in N \mid \exists v \in N, (u, v) \in S \vee (v, u) \in S\}$).

Un *sous-graphe commun* est un graphe isomorphe à des sous-graphes de G et G'. La similarité de deux graphes est généralement définie par rapport à la taille d'un sous-graphe commun [3] : plus ce sous-graphe est grand, plus les graphes sont similaires. La taille d'un sous-graphe est définie différemment selon que l'on considère des sous-graphes induits ou partiels : un *plus grand sous-graphe partiel (MCPS)* est un sous-graphe partiel qui a le plus grand nombre d'arcs, tandis qu'un *plus grand sous-graphe induit (MCIS)* est un sous-graphe induit qui a le plus grand nombre de sommets. La figure 1 montre un exemple de MCPS et MCIS.

2.2 Approches complètes existantes pour rechercher des MCIS et MCPS

La plupart des approches complètes pour rechercher des plus grands sous-graphes sont basées sur une reformulation du problème en un problème de recherche de clique maximum dans un graphe de compatibilité (dont les nœuds correspondent à des couples de nœuds pouvant être appariés, et les arêtes à des paires de couples de nœuds qui sont compatibles) [1, 5, 12].

McGregor [10] propose une approche différente basée sur un principe de Séparation et Evaluation (*Branch and Bound*) : chaque nœud de l'arbre de recherche correspond à l'appariement de deux composants, et la fonction d'évaluation estime le nombre de composants pouvant encore être appariés de sorte que la branche courante est coupée dès lors que cette borne devient inférieure à la taille du plus grand sous-graphe commun trouvé jusqu'ici.

Conte *et al* [4] ont comparé ces deux approches dans un même cadre expérimental sur une grande base de graphes. Ils ont montré qu'aucune approche ne surpasse l'autre : la meilleure approche change selon les caractéristiques des graphes.

Vismara et Valery [17] montrent comment rechercher des MCIS et MCPS à l'aide de la programmation par contraintes. Ils considèrent des cas particuliers de ces deux problèmes, où le sous-graphe recherché doit être connexe, et ils introduisent pour cela une contrainte globale de connexité. Ils utilisent des contraintes binaires de différence pour assurer que l'appariement calculé soit injectif. Ils comparent l'approche basée sur la programmation par contraintes avec une approche basée sur la recherche de cliques, et ils montrent que la programmation par contraintes obtient de meilleurs résultats.

3 Modélisation des problèmes de recherche de MCIS et MCPS à l'aide de CSP souples

Dans cette section, nous introduisons deux problèmes de satisfaction de contraintes (CSP) souples qui modélisent respectivement les problèmes de recherche du MCIS et du MCPS pour deux graphes $G = (N, A)$ et $G' = (N', A')$. Ces deux modèles diffèrent principalement sur leurs variables : pour le MCIS, les variables sont associées aux nœuds de G, alors que pour

le MCPS, les variables sont associées aux arcs. Dans les deux cas, la valeur affectée à la variable associée à un composant (nœud ou arc) de G correspond au composant avec lequel il est apparié dans G'. Comme certains composants peuvent ne pas être appariés, nous introduisons une valeur joker $\perp$ qui désigne le fait qu'un composant n'est pas apparié. Ainsi, pour le MCIS, le domaine de chaque variable x_u associée à un nœud $u \in N$ est $D(x_u) = N' \cup \{\perp\}$ tandis que pour le MCPS, le domaine de chaque variable x_{uv} associée à un arc $(u, v) \in A$ est $D(x_{uv}) = A' \cup \{\perp\}$.

Dans les deux cas, il existe deux types de contraintes. Un premier ensemble de contraintes binaires assure que les relations de voisinage définies par les arcs sont conservées. Pour le MCIS, ces contraintes binaires assurent que les relations d'adjacence entre les nœuds appariés sont conservées : étant données deux variables x_u et x_v, respectivement associées à des noeuds u et v de G, nous définissons

$$C_{arc}(x_u, x_v) \equiv \quad (x_u = \perp) \vee (x_v = \perp) \vee \\ ((u, v) \in A \Leftrightarrow (x_u, x_v) \in A')$$

Pour le MCPS, ces contraintes binaires assurent que les relations d'incidence entre les arcs appariés sont conservées : étant données deux variables x_{uv} et x_{wy} respectivement associées aux arcs (u, v) et (w, y) de G, nous définissons

$$C_{arc}(x_{uv}, x_{wy}) \equiv \quad (x_{uv} = \perp) \vee (x_{wy} = \perp) \vee \\ R((u, v), (w, y), x_{uv}, x_{wy})$$

où R est un prédicat qui vérifie que (u, v) et (w, y) ont les mêmes relations d'incidence que les arcs de G' affectés à x_{uv} et x_{wy}, *i.e.*, $R((u, v), (w, y), (u', v'), (w', y')) \equiv (u = v \Leftrightarrow u' = v') \wedge (u = w \Leftrightarrow u' = w') \wedge (u = y \Leftrightarrow u' = y') \wedge (v = w \Leftrightarrow v' = w') \wedge (v = y \Leftrightarrow v' = y') \wedge (w = y \Leftrightarrow w' = y')$.

Enfin, nous devons exprimer le fait que l'appariement doit être injectif (deux composants différents de G doivent être appariés à deux composants différents de G'). Ce genre de contrainte pourrait être modélisée avec une contrainte globale *allDiffExcept⊥(X)* qui oblige toutes les variables de X à prendre des valeurs distinctes, sauf si elles sont affectées à la valeur joker $\perp$ [2]. Pour trouver un plus grand sous-graphe commun, nous cherchons un appariement partiel injectif qui maximise le nombre de composants appariés, *i.e.*, nous devons minimiser le nombre de variables affectées à $\perp$. Cela pourrait être réalisé par l'ajout d'une contrainte *atmost*$(b - 1, X, \perp)$ à chaque fois qu'une solution cohérente σ est trouvée, où b est le nombre de variables affectées à $\perp$ dans σ. Cependant, ce modèle sépare l'évaluation de la fonction de coût de la contrainte globale *allDiff*.

Les contraintes globales d'optimisation permettent généralement un meilleur filtrage en liant les contraintes aux variables de coût à optimiser, comme cela a été proposé dans [6]. En particulier, la contrainte souple *allDiff* [11] lie un ensemble X de variables à une variable additionnelle *cost*, qui est contrainte à être égale au nombre de variables de X dont on devrait changer la valeur pour satisfaire la contrainte globale *allDiff(X)*, et qui doit être minimisée (nous considérons que les coûts de violation sont basés sur les variables).

Pour trouver un appariement partiel injectif qui minimise le nombre de composants non appariés, nous introduisons une variable supplémentaire, $x_\perp$, dont le domaine est $D(x_\perp) = \{\perp\}$, et nous ajoutons une contrainte souple *allDiff*$(X \cup \{x_\perp\}, cost)$. Notons que $x_\perp$ est toujours affectée à $\perp$: cette variable garantit que toutes les autres variables sont affectées à des valeurs différentes de $\perp$ à chaque fois que cela est possible (*i.e.*, quand G et G' sont isomorphes).

Nous pouvons maintenant définir les deux CSP souples modélisants les problèmes de recherche de MCIS et de MCPS.

MCIS. Nous définissons le CSP :
- Variables : $X = \{x_u \mid u \in N\} \cup \{x_\perp, cost\}$
- Domaines : $D(x_\perp) = \{\perp\}$, $D(cost) = \{0, \ldots, |N|\}$ et, $\forall u \in N, D(x_u) = N' \cup \{\perp\}$
- Contraintes dures : $\forall \{u, v\} \subseteq N, C_{arc}(x_u, x_v)$
- Contrainte souple : *allDiff* $(X, cost)$

MCPS. Nous définissons le CSP :
- Variables : $X = \{x_{uv} \mid (u, v) \in A\} \cup \{x_\perp, cost\}$
- Domaines : $D(x_\perp) = \{\perp\}$, $D(cost) = \{0, \ldots, |A|\}$ et, $\forall (u, v) \in A, D(x_{uv}) = A' \cup \{\perp\}$
- Contraintes dures : $\forall \{(u, v), (w, y)\} \subseteq A, C_{arc}(x_{uv}, x_{wy})$
- Contrainte souple : *allDiff* $(X, cost)$

Calcul d'un plus grand sous-graphe commun à partir d'une solution d'un CSP souple. Une solution est une affectation σ des variables de X qui satisfait toutes les contraintes dures et qui minimise le coût de violation de la contrainte souple de sorte que $\sigma(cost)$ soit égal à ce coût de violation. Soit $\sigma(X) \setminus \perp$ l'ensemble des valeurs différentes de $\perp$ qui sont affectées à des variables de X. Nous pouvons facilement vérifier que, pour le MCIS (resp. MCPS), $G'_{\downarrow \sigma(X) \setminus \perp}$ est isomorphe à un sous-graphe induit (resp. partiel) de G.

Notons que nous ne pouvons pas définir le sous-graphe commun induit en gardant simplement chaque nœud de G dont la variable associée a été affectée à une valeur différente de $\perp$. En effet, quand plusieurs nœuds de G ont le même voisinage, il peut arriver que

les variables associées à ces nœuds soient affectées à la même valeur (différente de $\perp$).

Considérons par exemple les graphes de la figure 1. Pour le MCIS, l'affectation $\sigma = \{x_\perp = \perp, x_a = \perp, x_b = 3, x_d = 4, x_c = 2, x_f = 5, x_e = 4\}$ est une solution optimale. Dans ce cas, $\sigma(X) \setminus \perp = \{2, 3, 4, 5\}$ et $G'_{\downarrow\sigma(X)\setminus\perp}$ est le sous-graphe obtenu en supprimant le nœud 1 de G'. Dans cette solution, à la fois x_d et x_e sont affectées à 4 car d et e ont le même voisinage.

La taille du plus grand sous-graphe commun $G'_{\downarrow\sigma(X)\setminus\perp}$ est égale à $c - \sigma(cost)$ où c est le nombre de composants de G ($c = |N|$ pour le MCIS et $c = |A|$ pour le MCPS), et $\sigma(cost)$ est la valeur de la variable *cost* dans la contrainte souple *allDiff*. Comme la valeur de *cost* est minimale, la taille de $G'_{\downarrow\sigma(X)\setminus\perp}$ est maximale.

Sur notre exemple précédent, nous avons $\sigma(cost) = 2$ et $|N| = 6$ de sorte que $G'_{\downarrow\sigma(X)\setminus\perp}$ a 4 nœuds.

Extension aux graphes étiquetés. Dans les graphes étiquetés, les nœuds et les arcs sont associés à des étiquettes. Dans ce cas, le sous-graphe commun doit apparier des composants ayant des étiquettes identiques. Ce type de contrainte est facile à intégrer dans notre modèle. Pour le MCIS, nous limitons le domaine de chaque variable x_u aux nœuds qui ont la même étiquette que u, et nous assurons que les étiquettes des arcs sont préservées dans les contraintes C_{arc}. Pour le MCPS, nous limitons le domaine de chaque variable x_{uv} aux arcs qui ont la même étiquette que (u, v) et dont les extrémités ont les mêmes étiquettes.

4 Propagation de contraintes

Les deux CSP introduits dans la section précédente sont très similaires : ils combinent tous deux un ensemble de contraintes binaires dures avec une contrainte souple *allDiff*. Nous proposons de considérer différents niveaux de propagation de ces contraintes.

4.1 Propagation de la contrainte souple allDiff

Nous considérons 3 niveaux de propagation.

GAC(allDiff). La plus forte propagation, notée *GAC(allDiff)*, assure la cohérence d'arc généralisée comme cela est proposé dans [11]. Plus précisément, nous recherchons un couplage maximum dans le graphe biparti $G_b = (X, V, E_b)$ où X est l'ensemble des variables, V est l'ensemble des valeurs dans les domaines des variables, et E_b est l'ensemble des arêtes $(x, v) \in X \times V$ tel que $v \in D(x)$. Un couplage de G_b est un sous-ensemble d'arêtes de E_b dont tous les sommets sont différents. La cardinalité d'un couplage maximum de G_b donne le nombre maximum de variables qui peuvent être affectées à des valeurs différentes et, par conséquent, fournit une borne supérieure pour *cost*. Lorsque la taille du couplage maximum de G_b est supérieure à la borne inférieure de *cost*, nous ne pouvons pas filtrer les domaines des variables. Cependant, dès que le couplage maximum de G_b est aussi grand que la borne inférieure de *cost*, nous pouvons filtrer les domaines en cherchant les arêtes de G_b n'appartenant à aucun couplage maximum (voir [14, 11] pour plus de détails).

Comme proposé dans [11, 14, 8], nous utilisons l'algorithme de [9] pour calculer un couplage maximum, et nous exploitons le fait que cet algorithme est incrémental : à chaque nœud de l'arbre de recherche, nous mettons à jour le dernier couplage calculé en enlevant les arêtes correspondant aux valeurs filtrées, et nous complétons ce couplage jusqu'à ce qu'il devienne maximum.

bound(cost)+FC(diff). Nous avons également considéré un filtrage moins fort, noté bound(cost)+FC(diff). Ce filtrage n'assure pas la cohérence d'arc généralisée, mais vérifie simplement qu'il existe un couplage du graphe biparti G_b dont la taille est supérieure ou égale à la borne inférieure de *cost*. Cette vérification est faite de façon incrémentale et fainéante : à chaque nœud de l'arbre de recherche, après avoir mis à jour le dernier couplage calculé, nous ne cherchons à l'étendre que si sa taille est strictement inférieure à la borne inférieure de *cost*. Cette vérification de borne est combinée à une simple vérification en avant (*forward-checking*) de la décomposition binaire de la contrainte globale *allDiff* : à chaque fois qu'une variable est affectée à une valeur v différente de $\perp$, nous enlevons v des domaines des variables non affectées.

FC(diff). Enfin, la propagation la plus faible, notée FC(diff), est une simple vérification en avant de la décomposition binaire de la contrainte *allDiff* qui enlève la valeur v des domaines des variables à chaque fois que v est affectée à une variable. La borne supérieure de la variable *cost* est mise-à-jour à chaque fois qu'une variable est affectée à $\perp$.

4.2 Propagation des contraintes dures C_{arc}

Tant que le domaine de la variable *cost* n'a pas été réduit à un singleton par la propagation de la contrainte souple *allDiff*, la valeur joker $\perp$ appartient au domaine de chaque variable non affectée. Dans ce cas, une vérification en avant des contraintes dures C_{arc} assure la cohérence d'arc de ces contraintes. En

effet, pour chaque paire de variables non affectées (x_i, x_j), et pour chaque valeur $v \in D(x_i)$, la valeur $\perp$ appartient à $D(x_j)$ et est un support de v car $C_{arc}(x_i, x_j)$ est satisfaite dès lors que x_i ou x_j sont affectées à $\perp$.

Dès lors que le domaine de la variable *cost* a été réduit à un singleton, $\perp$ est supprimé des domaines des variables non affectées. Dans ce cas, le maintien de la cohérence d'arcs (MAC) peut permettre de filtrer plus de valeurs qu'une simple vérification en avant. Par conséquent, nous avons considéré deux niveaux de propagation :

- FC(C_{arc}) effectue une simple vérification en avant des contraintes C_{arc} ;
- MAC(C_{arc}) maintient la cohérence d'arc de ces contraintes (cependant, comme la vérification en avant assure la cohérence d'arc tant que $\perp$ n'a pas été supprimé des domaines, nous n'effectuons qu'une vérification en avant quand $\perp$ appartient aux domaines, et nous ne maintenons effectivement la cohérence d'arc qu'à partir du moment ou $\perp$ a été enlevé des domaines).

5 Résultats expérimentaux

Modèles comparés. Nous comparons les cinq modèles suivants :

- FC = FC(C_{arc}) + FC(diff) ;
- FC+bound = FC(C_{arc})+bound(cost)+FC(diff) ;
- FC+GAC = FC(C_{arc}) + GAC(allDiff) ;
- MAC+bound = MAC(C_{arc}) + bound(cost) + FC(diff) ;
- MAC+GAC = MAC(C_{arc}) + GAC(allDiff).

En terme de puissance de filtrage, nous avons les relations suivantes : FC < FC+bound < FC+GAC < MAC+GAC et FC+bound < MAC+bound < MAC+GAC. En revanche, FC+GAC et MAC+bound ne sont pas comparables.

Notons que le modèle FC correspond en fait à l'approche par séparation et évaluation proposée par McGregor dans [10], ainsi qu'au modèle proposé dans [17] (excepté le fait que, dans [17], une contrainte de connectivité est ajoutée afin de garantir que les sous-graphes trouvés sont bien connexes).

Pour les 5 modèles, nous utilisons l'heuristique d'ordre *minDom* pour le choix des variables, et les valeurs sont choisies par ordre croissant. Les 5 modèles ont été programmés en C.

Jeux d'essais. Nous considérons une base de graphes générés aléatoirement et décrits dans [4]. Pour chaque graphe de la base, il existe trois étiquetages différents tels que le nombre d'étiquettes différentes est égal à 33%, 50% ou 75% du nombre de nœuds.

Nous comparons les résultats expérimentaux pour trois jeux d'essais de difficultés croissantes :

- Le jeu d'essai 1 recherche des MCIS dans des graphes orientés et étiquetés tels que le nombre d'étiquettes différentes est égal à 33% du nombre de nœuds (quand on augmente ce ratio, le problème devient beaucoup plus facile).
- Le jeu d'essai 2 recherche également des MCIS, mais dans des graphes non orientés et non étiquetés. Dans ce cas, les domaines des variables sont plus grands (car tous les sommets sont compatibles du fait de l'absence d'étiquettes), et les contraintes C_{arc} sont moins fortes.
- Le jeu d'essai 3 recherche des MCPS dans des graphes orientés et non étiquetés. Dans ce cas, les domaines des variables contiennent tous les arcs de G' et sont donc de plus grande taille que dans les deux cas précédents.

Nous avons adapté la taille des graphes en fonction de la difficulté de ces jeux d'essai de sorte que les problèmes puissent être résolus dans une limite de temps acceptable : pour le jeu d'essai 1 (resp. 2 et 3), nous considérons des graphes ayant 40 (resp. 30 et 20) nœuds.

Pour chaque jeu d'essai, nous donnons les résultats obtenus pour différentes classes de graphes :

- des graphes connexes générés aléatoirement dont la connectivité est $\eta \in \{0.05, 0.2\}$ (r005 et r02) ;
- des maillages réguliers 2D, 3D et 4D (m2D, m3D et m4D) :
- des maillages irréguliers 2D, 3D et 4D, avec $\rho = 0.6$ (m2Dr, m3Dr, et m4Dr) ;
- des graphes réguliers dont le degré est borné, avec $V \in \{3, 9\}$ (b03 et b09) ;
- des graphes irréguliers dont le degré est borné, avec $V \in \{3, 9\}$ (b03m et b09m).

Chaque classe contient 150 paires de graphes correspondant aux 30 premières instances pour chacune des 5 tailles possibles de plus grand sous-graphes (*i.e.*, 10%, 30%, 50%, 70% et 90% du nombre de nœuds).

Le tableau 1 compare les 5 modèles sur les 3 jeux d'essais. Nous constatons que le modèle FC (correspondant aux approches proposées dans [10] et [17]) est clairement dépassé par tous les autres modèles qui effectuent des filtrages plus forts. En effet, FC effectue un bornage passif de la variable *cost*, en comptant simplement le nombre de variables devant être affectées à $\perp$. Tous les autres modèles ont un mécanisme de calcul de borne actif qui vérifie que le nombre de variables pouvant être affectées à des valeurs différentes est suffisamment grand.

Le calcul de bornes fainéant (bound(cost)) introduit dans la section 4 réduit de façon drastique l'espace de recherche et les temps de FC+bound sont toujours si-

		FC			FC+bound			FC+GAC			MAC+bound			MAC+GAC		
		%S	temps	#Kn	%S	temps	#Kn	%S	temps	#Kn	%S	temps	#Kn	%S	temps	#Kn
Jeu d'essai 1	b03	100	105.79	56074	100	20.81	6066	100	24.83	4831	100	**13.43**	3166	100	13.86	1502
	b03m	100	143.74	80297	100	26.17	7754	100	28.86	5651	100	16.91	4021	100	**15.06**	1742
	b09	100	0.11	50	100	0.07	19	100	0.08	17	100	**0.06**	15	100	0.08	10
	b09m	100	0.12	54	100	0.08	22	100	0.09	20	100	**0.07**	17	100	0.10	11
	m2D	100	98.62	53960	100	18.05	5200	100	20.19	3664	100	12.25	2876	100	**10.94**	1220
	m2Dr	100	8.06	3990	100	3.14	864	100	3.65	724	100	**2.21**	523	100	2.38	291
	m3D	100	15.05	7532	100	5.55	1536	100	5.82	1157	100	**3.69**	865	100	4.86	439
	m3Dr	100	3.90	1913	100	1.62	419	100	1.82	359	100	1.14	273	100	**1.13**	154
	m4D	100	97.33	50940	100	12.09	3147	100	12.94	2496	100	**8.17**	1832	100	9.28	835
	m4Dr	100	5.85	2745	100	1.87	471	100	2.04	387	100	**1.38**	325	100	1.50	169
	r005	100	19.47	10540	100	4.72	1295	100	5.68	1040	100	**3.17**	741	100	3.57	393
	r02	100	0.02	10	100	0.02	6	100	0.02	5	100	**0.01**	4	100	0.02	3
Jeu d'essai 2	b03	72	756.25	312080	100	**68.87**	10256	100	77.93	7728	97	212.41	3679	98	231.77	2301
	b03m	57	1081.52	441952	100	**101.77**	14749	100	121.99	12043	97	343.77	6017	97	397.14	4010
	b09	100	147.80	62050	100	**35.09**	7709	100	40.27	6699	100	41.49	6068	100	44.69	3531
	b09m	99	342.89	149613	100	**86.07**	20054	100	94.98	16364	100	101.07	15347	100	103.71	8439
	m2D	61	985.35	394241	100	**103.17**	16003	100	131.48	13532	96	365.66	7582	95	411.89	4491
	m2Dr	62	998.30	383680	100	**171.70**	29757	100	201.49	24344	99	428.35	17228	98	482.21	9128
	m3D	76	737.81	277429	100	**81.78**	13206	100	101.71	11570	100	240.58	7538	98	284.13	4331
	m3Dr	83	681.18	266386	100	**115.49**	21872	100	156.56	20579	100	254.37	13715	100	316.93	8262
	m4D	46	1276.08	498405	100	**120.71**	17386	100	129.53	13257	100	360.14	8699	96	423.83	5704
	m4Dr	50	1448.08	549375	100	**165.51**	27849	100	195.86	23127	100	421.02	16238	100	467.62	8966
	r005	50	1236.75	515935	99	**142.04**	22122	98	175.62	17625	93	443.26	8601	92	494.76	5099
	r02	100	474.12	222831	100	246.16	67044	100	283.70	58036	100	**238.91**	53792	100	242.84	32445
Jeu d'essai 3	b03	100	34.44	9364	100	8.67	1178	100	10.93	1163	100	**7.13**	582	100	8.37	392
	b03m	100	47.41	13809	100	9.62	1350	100	10.41	1172	100	7.33	700	100	**7.04**	386
	b09	0	-	-	0	-	-	0	-	-	0	-	-	0	-	-
	b09m	0	-	-	0	-	-	0	-	-	0	-	-	0	-	-
	m2D	100	214.00	59029	100	32.17	3933	100	33.69	3324	100	26.16	2159	100	**25.83**	1165
	m2Dr	0	-	-	0	-	-	0	-	-	0	-	-	0	-	-
	m3D	90	1122.39	263519	100	206.22	23399	100	282.27	24543	100	**187.90**	14170	100	226.57	9000
	m3Dr	0	-	-	0	-	-	0	-	-	0	-	-	0	-	-
	r005	100	11.28	4333	100	2.12	354	100	2.56	358	100	**1.39**	144	100	1.56	98
	r02	0	-	-	0	-	-	0	-	-	0	-	-	0	-	-

TABLE 1 – Comparaison des 5 modèles sur les 3 jeux d'essais. Chaque ligne donne successivement le nom de la classe et, pour chaque modèle, le pourcentage d'instances (%S) résolues en moins de 30mn sur un 2.26 GHz Intel Xeon E5520, le temps CPU en secondes (temps) et le nombre de milliers de nœuds de l'arbre de recherche (#Kn). Le temps CPU ainsi que le nombre de nœuds sont des résultats moyens sur les instances résolues uniquement.

gnificativement plus bas que ceux de FC. GAC(allDiff) réduit encore plus l'espace de recherche mais la différence est moins forte de sorte que les temps CPU de FC+GAC sont supérieurs à ceux de FC+bound. Cette tendance est observée sur les trois jeux d'essais.

Le fait de remplacer FC(C_{arc}) par MAC(C_{arc}) réduit fortement le nombre de nœuds explorés, mais ce filtrage a aussi un coût plus important de sorte qu'il ne permet pas toujours de réduire le temps CPU : il améliore les performances sur les jeux d'essais 1 et 3, mais les détériore sur le jeu d'essai 2. Ainsi, les meilleures approches sont MAC+bound et MAC+GAC pour les jeux d'essais 1 et 3 tandis que la meilleure approche est FC+bound pour le jeu d'essai 2. En fait, les jeux d'essais 1 et 3 (tels que les graphes sont orientés) sont plus contraints que ceux du jeu d'essai 2, ce qui permet à MAC de filtrer plus de valeurs.

6 Conclusion

Nous avons montré comment modéliser des problèmes de recherche de plus grands sous-graphes communs sous la forme de problèmes de satisfaction de contraintes mêlant des contraintes dures, permettant d'assurer que les relations d'adjacence définies par les arcs sont conservées dans le sous-graphe commun, à une contrainte *allDiff* souple, permettant de maximiser la taille du sous-graphe commun. Nous avons considéré deux variantes du problème : la recherche d'un plus grand sous-graphe commun induit, où il s'agit de maximiser le nombre de sommets du sous-graphe, et la recherche d'un plus grand sous-graphe partiel, où il s'agit de maximiser le nombre d'arcs du sous-graphe.

Nous avons également introduit un nouvel algorithme pour la propagation de la contrainte souple *allDiff* : cet algorithme n'assure pas la cohérence d'arc généralisée mais vérifie simplement que le nombre de

variables pouvant être affectées à des valeurs différentes est supérieur ou égal à la borne inférieure de la variable de coût associée à la contrainte souple *allDiff*. Cette vérification est faite de façon incrémentale et fainéante, ce qui permet une propagation beaucoup plus rapide que la cohérence d'arc généralisée. Les résultats expérimentaux montrent que cette propagation fainéante assure un bon compromis entre force de filtrage et temps de propagation.

Les travaux futurs concerneront l'étude d'heuristiques d'ordre pour ces problèmes. En effet, les heuristiques d'ordre sont généralement très importantes pour améliorer l'efficacité de la résolution de problèmes d'optimisation, l'objectif étant de guider la recherche vers les meilleures solutions en premier de façon à pouvoir couper plus de branches. Nous souhaitons également effectuer plus d'expérimentations, sur des graphes de différentes natures, afin d'identifier les techniques de propagation les mieux adaptées en fonction de la nature des graphes considérés.

Références

[1] E. Balas and C. S. Yu. Finding a maximum clique in an arbitrary graph. *SIAM Journal on Computing*, 15(4) :1054–1068, 1986.

[2] N. Beldiceanu, M. Carlsson, S. Demassey, and Thierry Petit. Global constraint catalog : Past, present and future. *Constraints*, 12(1) :21–62, 2007.

[3] H. Bunke and K. Sharer. A graph distance metric based on the maximal common subgraph. *Pattern Recognition Letters*, 19(3) :255–259, 1998.

[4] 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. *Graph Algorithms and Applications*, 11(1) :99–143, 2007.

[5] P. J. Durand, R. Pasari, J. W. Baker, and C. Tsai. An efficient algorithm for similarity analysis of molecules. *Internet Journal of Chemistry*, 2, 1999.

[6] F. Focacci, A. Lodi, and M. Milano. Cost-based domain filtering. In *CP'99*, 1999.

[7] M. R. Garey and D. S. Johnson. Computer and intractability. In *Freeman*, 1979.

[8] I. Gent, I. Miguel, and P. Nightingale. Generalised arc consistency for the alldiff constraint : An empirical survey. *Artificial Intelligence*, 172(18) :1973–2000, 2008.

[9] J. E. Hopcroft and R. M. Karp. An n5/2 algorithm for maximum matchings in bipartite graphs. *SIAM Journal on Computing*, 2(4) :225–231, 1973.

[10] J. J. McGregor. Backtrack search algorithms and the maximal common subgraph problem. *Software Practice and Experience*, 12(1) :23–34, 1982.

[11] T. Petit, J.-C. Régin, and C. Bessière. Specific filtering algorithms for over-constrained problems. In *CP'01, LNCS 2239*, pages 451–464. Springer, 2001.

[12] J. W. Raymond, E. J. Gardiner, and P. Willett. Calculation of graph similarity using maximum common edge subgraphs. *The Computer Journal*, 45(6), 2002.

[13] J. W. Raymond and P. Willett. Maximum common subgraph isomorphism algorithms for the matching of chemical structures. *Journal of computeraided molecular design*, 16(7) :521–533, 2002.

[14] J.-C. Régin. A filtering algorithm for constraints of difference in csps. In *AAAI-94*, pages 362–367, 1994.

[15] J.-C. Régin. *Développement d'Outils Algorithmiques pour l'Intelligence Artificielle. Application à la Chimie Organique.* PhD thesis, Université Montpellier II, 1995.

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

[17] P. Vismara and B. Valery. Finding maximum common connected subgraphs using clique detection or constraint satisfaction algorithms. *Communications in Computer and Information Science*, 14(1) :358–368, 2008.

[18] S. Zampelli, Y. Deville, and C. Solnon. Solving subgraph isomorphism problems with constraint programming. *Constraints*, 15(3) :327–353, 2010.