Recherche d'un plus grand sous-graphe commun par décomposition du graphe de compatibilité

Maël Minot[1,2] **Samba Ndojh NDIAYE**[1,3] **Christine SOLNON**[1,2]

[1] Université de Lyon - LIRIS
[2] INSA-Lyon, LIRIS, UMR5205, F-69621, France
[3] Université Lyon 1, LIRIS, UMR5205, F-69622 France

{mael.minot,samba-ndojh.ndiaye,christine.solnon}@liris.cnrs.fr

Résumé

La taille d'un plus grand sous-graphe commun permet de mesurer la similarité entre des objets représentés par des graphes. Cependant, trouver un tel sous-graphe est un problème $\mathcal{NP}$-difficile. Nous décrivons dans cet article une méthode de décomposition, basée sur le graphe de compatibilité, qui produit des sous-problèmes indépendants. Des résultats expérimentaux montrent l'efficacité de cette méthode sur des instances difficiles.

Abstract

The size of a maximum common induced subgraph serves as a measure to evaluate the similarity level between objects represented by graphs. However, finding such a subgraph is an $\mathcal{NP}$-hard problem. In this article, we describe a decomposition method, based on the compatibility graph, that produces independent subproblems. Experiments show that this method obtains very good results on difficult instances.

1 Introduction

La recherche d'un plus grand sous-graphe commun est un problème $\mathcal{NP}$-difficile particulièrement complexe qui n'en reste cependant pas moins intéressant du fait de ses nombreuses applications en chimie, biologie ou encore en traitement d'images, lorsqu'il est nécessaire de mesurer la similarité d'objets représentés par des graphes. Il existe deux principaux types d'approches complètes permettant de résoudre ce problème. La technique dite du *branch and bound* construit incrémentalement tous les sous-graphes communs possibles [16]. Une construction de sous-graphe est alors stoppée dès que l'algorithme parvient à prouver que ce sous-graphe ne permettra pas de construire une solution plus grande que la meilleure trouvée jusqu'alors. Les améliorations de cette technique impliquent généralement de trouver des moyens d'éliminer des branches plus tôt en prouvant leur inutilité. La Programmation par Contraintes (PPC, ou CP en anglais), notamment, permet de réduire le nombre de branches en propageant des contraintes [18, 26]. La PPC est compétitive avec l'état de l'art pour résoudre ce problème, mais il reste difficile de résoudre des instances dont les graphes ont plus de quelques dizaines de sommets. La seconde approche complète pour la recherche d'un plus grand sous-graphe commun est basée sur une reformulation du problème en un graphe dit *de compatibilité* [1, 2, 8, 13]. Chaque clique présente dans ce graphe de compatibilité correspond à un sous-graphe commun, et une clique *maximum* est un sous-graphe commun *maximum*. Ainsi, il devient possible de se contenter de résoudre le problème de la clique maximum dans le graphe de compatibilité.

Afin d'améliorer le processus de résolution, le problème peut être décomposé en sous-problèmes indépendants, qui offrent donc la possibilité d'être résolus en parallèle. Une telle approche a été présentée par exemple dans [7, 14, 15] pour le problème de la clique maximum, ou dans [21, 22] pour les problèmes de satisfaction de contraintes. Dans la plupart des cas, cette décomposition prend la forme d'un découpage des domaines. Par exemple, Régin et al. proposent dans [21, 22] de créer p sous-problèmes indépendants en choisissant k variables telles que la taille du produit cartésien de leurs domaines est proche de p. Un sous-

problème est alors généré pour chaque instanciation possible de ces k variables, excepté pour les instanciations provoquant d'emblée une inconsistance vis-à-vis des contraintes du problème (« Non Detected Inconsistent subproblems »), étant donné qu'il s'agit de branches qui n'auraient pas été considérées par une recherche séquentielle.

Un autre moyen de décomposer un problème est de tirer parti de sa structure [11, 5, 6]. Ces méthodes impliquent généralement une décomposition arborescente du graphe de contraintes. Cependant, dans le cas du problème du plus grand sous-graphe commun, les associations de sommets doivent être strictement distinctes, ce qui induit l'utilisation de contraintes *allDifferent* dans les modèles de PPC. La portée de ces contraintes est égale à l'ensemble des variables dans son entier, ce qui rend inutile toute méthode basée sur une décomposition arborescente du fait de l'apparition d'un sous-problème équivalent au problème initial, non-décomposé. Une variante proposée par [3] se base sur la microstructure du problème de satisfaction de contraintes plutôt que sur son graphe de contraintes. Cette nouvelle approche produit des problèmes complètement indépendants, ce qui permet de conserver la possibilité de les résoudre en parallèle.

Dans cet article, nous appliquons cette technique au problème du plus grand sous-graphe commun, afin de le décomposer en un ensemble de problèmes indépendants tout en réduisant la taille de l'espace de recherche à explorer. Ces travaux sont la continuation de ceux présentés dans [17].

La section qui suit présente le problème du plus grand sous-graphe commun de manière plus détaillée, ainsi que quelques notions de PPC. Une méthode de décomposition destinée aux problèmes de PPC sera également résumée. La section 3 explique comment adapter cette méthode de décomposition au problème du plus grand sous-graphe commun. La section 4 donne les premiers résultats expérimentaux et les analyse, introduisant ainsi la section 5, qui propose des voies d'amélioration des performances. Les résultats finaux ainsi obtenus sont quant à eux présentés en section 6.

2 Préliminaires

2.1 Plus grand sous-graphe commun

Définition 1 *Un graphe* non orienté G *est défini par un ensemble fini de nœuds* N_G *et un ensemble* $E_G \subseteq N_G \times N_G$ *d'arêtes. Chaque arête est un couple non orienté de nœuds.*
Dans un graphe orienté, E_G *est un ensemble d'*arcs, *qui sont des couples* orientés *de nœuds.*

Dans les définitions suivantes, nous nous plaçons dans le cadre des graphes *orientés*. Cependant, les notions abordées peuvent être étendues aux graphes non orientés.

Définition 2 *Soient deux graphes* G *et* G'. G *est* isomorphe *à* G' *s'il existe une bijection* $f : N_G \rightarrow N_{G'}$ *préservant les arêtes,* i.e., $\forall(u,v) \in N_G \times N_G, (u,v) \in E_G \Leftrightarrow (f(u),f(v)) \in E_{G'}$.

Définition 3 G' *est un sous-graphe induit de* G *si* $N_{G'} \subseteq N_G$ *et* $E_{G'} = E_G \cap (N_{G'} \times N_{G'})$. *On dit alors que* G' *est le sous-graphe de* G *induit par le sous-ensemble de nœuds* $N_{G'}$, *et on note* $G' = G_{\downarrow N_{G'}}$.

En d'autres termes, G' est obtenu à partir de G en retirant tout nœud de G qui ne se trouve pas dans $N_{G'}$, et en conservant uniquement les arêtes dont *les deux extrémités* sont présentes dans $N_{G'}$.

Les sous-graphe dits *partiels* sont, eux, obtenus en considérant un sous-ensemble $N_{G'}$ de N_G et un sous-ensemble d'arêtes de G dont les extrémités sont toutes deux dans $N_{G'}$, comme formalisé dans la définition suivante :

Définition 4 G' *est un sous-graphe* partiel *de* G *si* $N_{G'} \subseteq N_G$ *et* $E_{G'} \subseteq E_G \cap (N_{G'} \times N_{G'})$. *Un sous-graphe partiel* G' *de* G *induit par un sous-ensemble* $E_{G'}$ *de* E_G *est défini par* $(N_{G'},E_{G'})$, *où* $N_{G'} = \cup_{(x,y)\in E_{G'}}(x,y)$.

Définition 5 *Un* sous-graphe commun *de deux graphes* G *et* G' *est un graphe qui est isomorphe à des sous-graphes de* G *et* G'.

Définition 6 *Un plus grand sous-graphe induit commun (ou « MCIS » pour* « Maximum Common Induced Subgraphe »*) est un sous-graphe induit commun avec un nombre de nœuds maximal.*

Définition 7 *Un plus grand sous-graphe partiel commun (ou « MCPS » pour* « Maximum Common Partial Subgraphe »*) est un sous-graphe partiel commun avec un nombre d'arêtes maximal.*

La figure 1 expose des exemples de MCIS et de MCPS.

Dans cet article, nous nous restreignons à la recherche d'un MCIS. Cependant, les résultats peuvent aisément être étendus au problème du MCPS, comme le montre [18].

2.2 Modèle de PPC pour le problème du MCIS

Définition 8 *Un problème de satisfaction de contraintes (« CSP » pour* « Constraint Satisfaction Problem »*) est défini par un triplet* (X, D, C) *: X est*

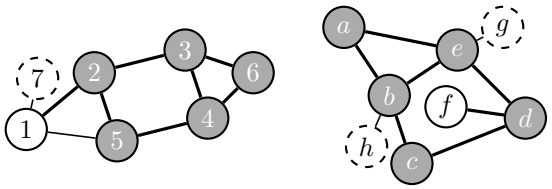

FIGURE 1 – Les nœuds pleins de ces deux graphes forment un MCIS. Une bijection peut être définie de la manière suivante : $2 \leftrightarrow d$, $3 \leftrightarrow e$, $4 \leftrightarrow b$, $5 \leftrightarrow c$, $6 \leftrightarrow a$. Les nœuds dont le contour n'est *pas* en pointillés, associés aux arêtes épaisses, forment quant à eux un MCPS.

un ensemble fini de variables, D associe un domaine $D(x_i)$ à chaque variable $x_i \in X$, et C est un ensemble de contraintes. Chaque contrainte est définie sur un sous-ensemble de variables et donne la liste des valeurs qui peuvent être affectées à ces variables simultanément. Une solution est une affectation de toutes les variables qui satisfait toutes les contraintes.

[18] a introduit un modèle CSP pour le problème du MCIS se montrant plus performant que le *branch and bound* standard de [16]. Puisque le problème du MCIS est un problème d'optimisation, ce modèle utilise la généralisation des CSP connue sous le terme CSP *soft*. Dans un CSP souple, les solutions doivent satisfaire un ensemble de contraintes dites *dures* tout en optimisant une *fonction objectif*, qui correspond au coût de violation associé à une combinaison de contraintes *souples* (*soft*). Pour deux graphes G et G' donnés, ce modèle est :

— Variables : $X = \{x_u \mid u \in N_G\} \cup \{x_\perp, cost\}$
— Domaines : $D(x_\perp) = \{\perp\}$, $D(cost) = [0,|N_G|]$, et $\forall u \in N_G, D(x_u) = N_{G'} \cup \{\perp\}$
— Contraintes dures : $\forall \{u,v\} \subseteq N_G, (x_u = \perp) \vee (x_v = \perp) \vee ((u,v) \in E_G \Leftrightarrow (x_u,x_v) \in E_{G'})$
— Contraintes souples : *softAllDiff*$(X,cost)$

Chaque nœud u de G est associé à une variable x_u dont le domaine contient tous les nœuds de G'. Les domaines contiennent également une valeur supplémentaire $\perp$ qui est utilisée lorsque u n'est associée à aucun nœud de G'. Une variable additionnelle $x_\perp$ joue un rôle particulier : elle reçoit forcément l'unique valeur de son domaine, qui n'est autre que $\perp$. Grâce à cette variable et à la contrainte souple *softAllDiff*, chaque autre variable se verra affecter une valeur différente de $\perp$ à chaque fois que cela sera possible. La variable *cost* est contrainte d'être égale au nombre de variables qui devraient changer de valeur pour satisfaire une hypothétique contrainte *allDiff* dure. L'ensemble de contraintes dures garantit la préservation des arêtes dans le sous-graphe commun construit. Une solution de ce CSP est une affectation de *toutes* les variables de X qui satisfait chaque contrainte dure tout en minimisant la valeur de *cost*. Le MCIS associé à une solution de ce CSP souple est défini par l'ensemble de nœuds $u' \in N_{G'}$ tel qu'il existe une variable $x_u \in X$ qui est associée à u' dans cette solution. [18] évalue expérimentalement différentes techniques de propagation de contraintes. La combinaison « MAC+Bound » fourni généralement de très bons résultats : *« Maintaining Arc Consistency »* (MAC) [23] est utilisée pour propager les contraintes dures ; *« Bound »* vérifie qu'il est toujours possible d'affecter des valeurs distinctes à suffisamment de variables de X pour faire mieux que le plus faible coût (valeur de *cost*) rescencé parmi les sous-graphes communs déjà trouvés (c'est une version allégée de GAC(*softAllDiff*) [19] qui calcule le nombre maximal de variables qui pourront obtenir des valeurs distinctes).

2.3 Graphe de compatibilité

Une autre approche visant à résoudre le problème du MCIS est basée sur sa reformulation en instance du problème de la *clique maximum* dans un *graphe de compatibilité* [1, 8, 20].

Définition 9 *Une* clique *est un sous-graphe dont les nœuds sont tous liés deux à deux. Une clique est dite* maximale *si elle n'est pas strictement incluse dans une autre clique, et* maximum *s'il s'agit d'une des plus grandes cliques du graphe considéré, en prenant comme critère le nombre de nœuds qui la composent.*

Définition 10 *Un graphe de compatibilité pour le problème du MCIS entre deux graphes G et G' est un graphe G_C dont l'ensemble des nœuds correspond à $N_{G_C} = N_G \times N_{G'}$ et où deux nœuds sont reliés par une arête s'ils sont* compatibles. *Soient (u,u') et (v,v') deux nœuds de G_C (i.e. $\{u,v\} \subseteq N_G$ et $\{u',v'\} \subseteq N_{G'}$). (u,u') et (v,v') sont compatibles si $u \neq v$ et $u' \neq v'$, et si les arêtes sont préservées (i.e. $(u,v) \in E_G \Leftrightarrow (u',v') \in E_{G'}$).*

Comme chacun pourra le constater dans la figure 2, une clique dans G_C décrit un ensemble d'associations compatibles entre les nœuds de G et de G'. Ainsi, une telle clique correspond à un sous-graphe induit commun, et une clique maximum dans G_C est un MCIS de G et G'. Il suit logiquement que toute méthode capable de trouver une clique maximum dans un graphe peut être employée pour résoudre le problème du MCIS.

2.4 TR-décomposition

[10] a introduit une méthode de décomposition, nommée TR-décomposition, visant à résoudre une instance de CSP binaire en divisant les domaines des

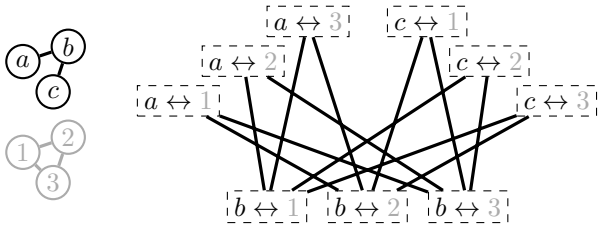

FIGURE 2 – Deux graphes et le graphe de compatibilité associé, qui souligne les combinaisons possibles d'affectations, par paires. Par exemple, puisque a et c ne sont reliés par aucune arête tandis que 1 et 3 le sont, les affectations $a \leftrightarrow 1$ et $c \leftrightarrow 3$ ne sont *pas* compatibles, car une arête serait amenée à disparaître, ce qui est formellement interdit par le problème du MCIS.

variables afin d'obtenir des sous-problèmes indépendants. Cette méthode s'appuie sur une réduction du CSP en une instance du problème de la recherche d'une clique de taille n (où n est le nombre de variables du CSP) dans la microstructure du CSP. La microstructure d'un CSP est un graphe similaire au graphe de compatibilité décrit en section 2.3. Chaque nœud de la microstructure correspond à une affectation, donc un couple $(variable, valeur)$. Deux nœuds sont reliés par une arête si et seulement si les affectations correspondantes sont compatibles, i.e. si chaque contrainte du problème est satisfaite lorsque ces deux affectations sont effectuées simultanément. De ce fait, une clique à n sommets dans la microstructure constitue une solution au CSP.

L'idée principale derrière la TR-décomposition est de trianguler la microstructure afin de générer plus aisément les sous-problèmes : un graphe quelconque peut comporter un nombre exponentiel de cliques maximales en fonction du nombre de nœuds. Un graphe triangulé, en revanche, ne peut avoir plus de $|N_G|$ cliques maximales, et la structure spécifique de ces graphes facilite également l'énumération de ces cliques [9]. [10] considère la classe des graphes triangulés afin de faire bénéficier la TR-décomposition de ces propriétés.

Définition 11 *Un graphe G est triangulé si et seulement si chacun de ses cycles de longueur supérieure ou égale à* 4 *possède au moins une* corde. *On appelle corde d'un cycle toute arête dont les extrémités sont deux nœuds* non consécutifs *du cycle.*

On appelle *triangulation* l'opération qui consiste à transformer un graphe donné en graphe triangulé *uniquement en ajoutant des arêtes.* La figure 3 montre une triangulation possible du graphe de compatibilité déjà rencontré dans la figure 2.

Puisque la triangulation ajoute des arêtes sans jamais en enlever, les cliques maximales du graphe

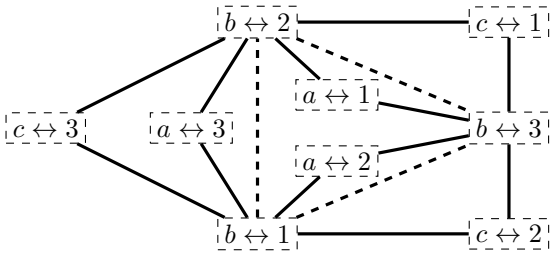

FIGURE 3 – Une version triangulée de graphe de compatibilité de la figure 2. Les arêtes qui ont dû être ajoutées (*« fill edges »*) sont indiquées en pointillés.

d'origine restent présentes au cours de la transformation : chacune est incluse dans une clique maximale du graphe triangulé. L'étape suivante de la TR-décomposition consiste à extraire ces cliques se trouvant au sein des nouvelles cliques maximales. Cette tâche est généralement plus aisée lorsque les cliques maximales du graphe triangulé sont petites, et donc plus proches des cliques d'origine.

En résumé, la méthode appelée TR-décomposition consiste à construire la microstructure du problème, à la trianguler, à extraire les cliques maximales de la microstructure triangulée, et à résoudre séparément les sous-problèmes induits par ces cliques, ce qui permet de retrouver les cliques de la microstructure d'origine.

3 Méthode de décomposition pour le problème du MCIS

Nous proposons ici de combiner la TR-décomposition avec le modèle de PPC défini dans [18] afin de résoudre le problème du MCIS en utilisant le graphe de compatibilité. Tandis que la TR-décomposition avait initialement été définie pour des instances de CSP, le modèle de PPC de [18] est un CSP *souple*. Une adaptation est donc nécessaire. Il se trouve que le problème du MCIS peut être résolu en trouvant une clique maximum dans le graphe de compatibilité. Nous inspirant de [10], nous suggérons d'utiliser la classe des graphes triangulés afin de tirer profit de leur nombre de cliques maximales limité.

Dans ce but, nous construisons le graphe de compatibilité de l'instance de problème du MCIS et triangulons ce graphe, ajoutant donc des arêtes. Ces arêtes, appelées *« fill edges »*, ajoutent des compatibilités erronées, car elles sont en contradiction avec la définition du graphe de compatibilité. De ce fait, une clique maximum dans ce nouveau graphe n'est pas forcément la meilleure solution. Ce n'est qu'un *sous-problème* – parmi d'autres – dans lequel il est possible que nous trouvions plusieurs solutions. Une clique maximum du

graphe de compatibilité reste une clique (mais pas forcément maximum ni même maximale) dans le graphe triangulé. De plus, cette clique apparaîtra forcément dans au moins une des cliques maximales de ce graphe triangulé. Pour ces raisons, chaque clique maximale du graphe triangulé définit un sous-problème (par le biais du sous-graphe qu'elle induit dans le graphe de compatibilité d'origine) dans lequel il est possible de trouver une clique maximum du graphe de compatibilité initial. De tels sous-problèmes peuvent être vus comme des instances habituelles du problème du MCIS pouvant être résolues en appliquant le modèle de PPC de [18], à la différence près que les domaines des variables sont réduits.

Plus précisément, pour chaque clique maximale K relevée dans la version triangulée du graphe de compatibilité associé à G et G', on définit un sous-problème ayant les mêmes variables et contraintes (décrites en section 2.2) que le problème initial. Les domaines, en revanche, sont restreints à

$$D(x_u) = \{v \in N_{G'} \mid (u \leftrightarrow v) \in K\} \cup \{\bot\}$$

Les domaines respectifs de $x_\bot$ et de *cost* restent quant à eux inchangés. Ainsi, par exemple, dans la figure 3, le sous-ensemble de nœuds $\{(b \leftrightarrow 1), (b \leftrightarrow 3), (c \leftrightarrow 2)\}$ est une clique maximale. Dans le sous-problème associé, $D(b) = \{1,3,\bot\}$, $D(c) = \{2,\bot\}$, et $D(a) = \{\bot\}$.

Chaque solution à un des sous-problèmes correspond à une clique maximale du graphe de compatibilité, et donc à un sous-graphe induit commun. Le sous-problème de l'exemple précédent possède deux solutions : associer b à 1 et c à 2, ou associer b à 3 et c à 2. Ces solutions correspondent à autant de cliques maximales du graphe de compatibilité : $\{(b \leftrightarrow 1), (c \leftrightarrow 2)\}$ et $\{(b \leftrightarrow 3), (c \leftrightarrow 2)\}$. L'arête $\{(b \leftrightarrow 1), (b \leftrightarrow 3)\}$ n'est pas utilisée pour construire ces cliques-solutions car il s'agit d'une *fill edge*, ajoutée lors du processus de triangulation et n'étant donc pas présente dans le graphe de départ.

Notez que ces sous-problèmes sont *complètement indépendants*, une caractéristique qui permet une implémentation parallèle extrêmement aisée du processus de résolution. Toute méthode capable de résoudre un CSP ou le problème de la clique maximum peut être employée pour résoudre ces sous-problèmes.

3.1 Bornes inférieures

Au cours de la résolution des sous-problèmes (que celle-ci se déroule en séquentiel ou en parallèle), il est possible d'ignorer tout sous-problème au sujet duquel on peut démontrer qu'il ne contient pas de solutions plus grandes que la meilleure trouvée jusque-là. En effet, certains sous-problèmes comportent des domaines contenant uniquement la valeur $\bot$. Une variable ayant un tel domaine n'aura d'autre choix que de se voir affecter cette valeur spéciale. Ainsi, si le nombre de variables ayant un domaine contenant autre chose que $\bot$ est plus petit que la taille du plus grand sous-graphe induit commun trouvé jusque-là, le sous-problème peut être ignoré. Par exemple, dans la figure 3, résoudre le sous-problème correspondant à la clique maximale $\{(b \leftrightarrow 1), (b \leftrightarrow 3), (c \leftrightarrow 2)\}$ fournit un MCIS potentiel à deux nœuds. Une fois cette solution trouvée, il est inutile d'examiner les sous-problèmes dans lesquels seules deux variables ou moins ont un domaine n'étant *pas* restreint à $\bot$: de tels sous-problèmes ne peuvent contenir de solutions comportant plus de deux nœuds et ne nous permettront donc pas de faire mieux que la solution précédemment trouvée.

Bien entendu, il est intéressant de trouver de grands sous-graphes induits communs tôt, car cela permet d'éliminer davantage de sous-problèmes. Pour ce faire, il est possible d'exécuter un algorithme heuristique afin d'obtenir rapidement une première solution, non optimale mais satisfaisante, avant de procéder à la décomposition. Dans nos expériences, nous utilisons *AntClique* [25] (disponible à l'adresse [24]) afin de rechercher des cliques maximales dans le graphe de compatibilité initial. Il est apparu que cette heuristique était capable de trouver une solution optimale pour plus de 90 % des instances résolues, malgré un temps d'exécution souvent négligeable par rapport au temps complet de résolution. Notez qu'*AntClique* ne prouve pas l'optimalité ; cette tâche est laissée à nos algorithmes de résolution de CSP.

Éliminer des sous-problèmes de cette manière était particulièrement important dans l'approche présentée par [3], car le CSP considéré était strict. Tout sous-problème avec au moins une variable au domaine vide pouvait être ignoré, car seules les solutions affectant une valeur à *chaque* variable étaient valides. Dans notre cas (qui est un problème d'optimisation), en revanche, les solutions valides peuvent se contenter de ne donner une valeur qu'à un sous-ensemble des variables. La seule manière d'éliminer un sous-problème en ayant la certitude de ne pas nuire à la complétude de l'approche est donc de prouver qu'il ne comporte pas de solution plus grande que la meilleure solution actuelle. Dans de telles conditions, il faut qu'un nombre significatif de valeurs aient un domaine vide. De plus, il se trouve que décomposer ce problème en se contentant de trianguler donne rarement lieu à de telles possibilités d'éliminations, dans les expériences que nous menons.

3.2 Complexité

Soient n et n' les nombres respectifs de nœuds des deux graphes dans lesquels nous recherchons un MCIS.

La première étape consiste à construire le graphe de compatibilité, chose faite en temps quadratique par rapport à son nombre de nœuds, i.e., en $\mathcal{O}((nn')^2)$. Nous lançons ensuite *AntClique* afin d'obtenir une borne inférieure de la taille du MCIS. Cet algorithme demande un temps en $\mathcal{O}((nn')^2)$, car le nombre de cycles d'*AntClique* est borné. La troisième partie de la résolution est la triangulation du graphe de compatibilité. Le but est ici d'ajouter le moins d'arêtes possible. Il se trouve qu'il s'agit d'un problème $\mathcal{NP}$-difficile. Nous sommes donc contraints d'utiliser une heuristique, en l'occurrence l'algorithme glouton *minFill* [12], qui nous permet d'effectuer cette triangulation en $\mathcal{O}((nn')^3)$. La quatrième étape est l'extraction des cliques maximales du graphe triangulé. Il y en a au plus nn', et chacune d'entre elles est obtenue en temps linéaire. Enfin, la dernière étape est de résoudre les sous-problèmes représentés par ces cliques maximales. L'initialisation du problème en fonction des sommets composant la clique demande un temps linéaire par rapport au nombre d'arêtes de cette clique. Chaque sous-problème a $\mathcal{O}(n)$ variables. Cependant, la taille des domaines dépend, elle, de la structure du graphe de compatibilité triangulé. Soit d la taille du plus grand domaine d'un sous-problème (avec $d \leq n' + 1$). Chaque sous-problème est résolu avec MAC+Bound, en $\mathcal{O}(d^3.n^2.d^n)$. Il en suit que la complexité temporelle globale de la TR-décomposition est en $\mathcal{O}((nn')^3 + nn'(d^3.n^2.d^n))$.

En comparaison, la complexité du problème du MCIS sans décomposition est en $\mathcal{O}(d'^3.n^2.d'^n)$ où d' est la taille du plus grand domaine dans le problème initial. Il faut noter que $d \leq d'$ et que selon la qualité de la décomposition la valeur de d peut être beaucoup plus petite que celle d'.

4 Premières expériences

Les expériences suivantes ont pour but de comparer l'approche de PPC pure de [18] (*MAC+Bound*) avec la TR-décomposition. Ces deux méthodes utilisent la même heuristique *AntClique* pour obtenir une solution initiale, qui sert de première borne inférieure.

Nos algorithmes ont été développés en C et compilés avec GCC, en utilisant le niveau d'optimisation `-O3`. Les programmes ont été exécutés avec une limite de temps de trois heures sur des processeurs Intel® Xeon® CPU E5-2670 0 à 2,60 GHz, avec 20 480 Ko de mémoire cache et 4 Go de RAM.

Les instances utilisées proviennent du *benchmark* présenté par [4] et ont été prises dans leur version étiquetée et orientée. Le nombre d'étiquettes différentes, pour une instance donnée, a été fixé à 15 % du nombre de nœuds des deux graphes de l'instance. Par exemple, quand les graphes ont 60 nœuds, il y a $60 \times 15 \div 100 = 9$ types d'étiquettes différents. ce paramètre permet de régler la difficulté des instances. Ces étiquettes sont prises en compte comme expliqué dans [18], en s'assurant que des nœuds associés pour construire un sous-graphe commun possèdent la même étiquette. De manière similaire, les arêtes situées entre des paires de nœuds correspondants doivent avoir la même étiquette. Chaque domaine est ainsi réduit aux valeurs associées à des nœuds ayant la même étiquette que le nœud associé à la variable.
Nous considérons trois types d'instances :

bvg *Bounded Valence Graphs*, dans lesquels les degrés des sommets ne peuvent dépasser une valeur fixe, propre au graphe.

rand *Random graphs*, dans lesquels les arêtes sont disposées de manière aléatoire (voir [4] pour plus de détails).

m2D, m3D and m4D *Meshes* (maillages) à deux, trois ou quatre dimensions respectivement.

Ces graphes ont un nombre de nœuds allant de 10 à 100 inclus. Nous considérons 3 268 instances prises au hasard parmi les 81 700 que présente le *benchmark*, sans favoriser l'un des types d'instances. Leurs proportions sont ainsi relativement préservées.

Puisque notre méthode de décomposition est avant tout conçue pour les instances particulièrement difficiles, nous nous concentrons sur les (nombreuses) instances que la méthode standard (*MAC+Bound* [18]) ne parvient pas à résoudre en moins de 100 secondes. Le nombre d'instances passe ainsi de 3 268 à seulement 1 177.

Le tableau 1 donne les spécificités des instances se trouvant dans les séries que nous avons utilisées, et ce avant et après le filtrage retirant les instances jugées trop faciles selon le critère susmentionné. On remarque que les graphes aléatoires (*rand*) sont moins présents après filtrage, ce qui signifie qu'ils ont tendance à être plus faciles. A contrario, les maillages représentent une part bien plus importante de notre panel d'instances après ce filtrage. Cela n'a rien d'une surprise, car les maillages sont connus pour rendre difficile la recherche d'un plus grand sous-graphe commun, et ceux-ci ont été créés par [4] principalement pour cette raison. Cette difficulté provient de la structure répétitive de ces graphes, qui fait apparaître de nombreux grands sous-graphes communs qui ne sont cependant que rarement optimaux.

Nous allons dans un premier temps évaluer la qualité

Tableau 1 – La proportion représentée par chaque type de graphe dans l'ensemble d'instances de départ, puis après exclusion des instances résolues en moins de 100 secondes sans décomposition.

	Toutes les instances	Instances difficiles
Total	3 268	1 177
bvg	1 232 (37,7 %)	394 (33,5 %)
rand	744 (22,8 %)	135 (11,5 %)
m2D	700 (21,4 %)	335 (28,5 %)
m3D	384 (11,8 %)	201 (17,1 %)
m4D	208 (6,4 %)	112 (9,5 %)

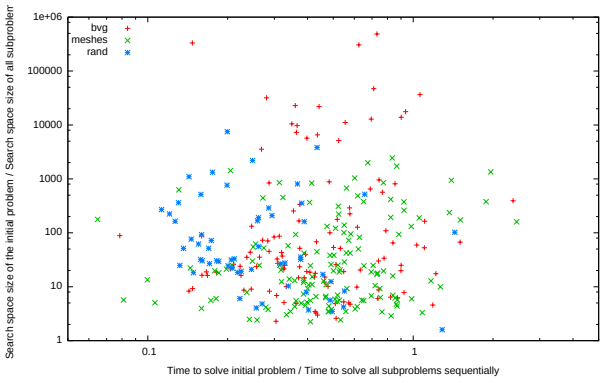

Figure 4 – La réduction de la taille de l'espace de recherche mise en relation avec la réduction du temps de calcul. Chaque point (x,y) correspond à une instance i. Soit S_i l'ensemble des sous-problèmes générés à partir de i. x donne le rapport entre le temps de calcul nécessaire à la résolution de i et celui nécessaire à la résolution de tous les sous-problèmes de S_i (sur un unique processeur, et en incluant le temps de décomposition). y donne le rapport entre l'espace de recherche de i et la somme des espaces de recherche des sous-problèmes de S_i.

de la décomposition par rapport à la taille des espaces de recherche avec et sans décomposition. La taille de l'espace de recherche d'un (sous-)problème est égale à la taille du produit cartésien de ses domaines, en comptant $\bot$. Dans le cadre de la TR-décomposition, la somme des tailles des espaces de recherche des différents sous-problèmes est une bonne mesure de la qualité de la décomposition dans le cas extrême où tous les sous-problèmes sont résolus consécutivement. Notre premier constat est que la TR-décomposition réduit l'espace de recherche. Nous étudions donc tout d'abord l'impact de cette réduction sur le processus de résolution. La figure 4 nous montre que cette réduction est d'un facteur généralement situé entre 2 et 10^6. Cependant, cette réduction ne se traduit pas systématiquement par une diminution du temps de calcul nécessaire à la résolution. Dans la figure 4, nous comparons le temps nécessaire à la résolution du problème initial et le temps nécessaire à la décomposition du problème et à la résolution de tous les sous-problèmes sur *un seul* processeur. Nous notons que seules 8 (resp. 10 et 2) instances de type *bvg* (resp. *meshes* et *rand*) sont résolues plus rapidement, dans ces conditions de processeur unique, avec la décomposition. Il s'agit d'un point négatif de notre approche, et nous comptons étudier cet inconvénient avec davantage d'attention.

Cependant, du fait que les sous-problèmes sont indépendants, nous pouvons les résoudre en parallèle plutôt que sur un unique processeur. On remarque que pour de nombreuses instances, il n'y a qu'un nombre très faible de gros sous-problèmes, entourés d'une multitude de sous-problèmes de taille réduite. De ce fait, il n'est pas extrêmement utile d'avoir autant de processeurs à notre disposition que nous avons généré de sous-problèmes. Si le nombre de processeurs est limité à m, nous pouvons, une fois la décomposition terminée, disposer les sous-problèmes dans une file (une méthode similaire à ce qui a été fait dans [21, 22]). Un sous-problème est initialement donné à chaque processeur. Par la suite, dès lors qu'un processeur a terminé la résolution du sous-problème qui lui avait été attribué, le sous-problème suivant dans la file lui est affecté. Pour améliorer la répartition de la charge de travail, nous tentons de placer les sous-problèmes semblant les plus complexes en premier dans la file, en utilisant la taille de l'espace de recherche comme critère d'estimation de la difficulté. Ainsi, la figure 5 montre que seuls 9 processeurs suffisent à obtenir des performances comparables avec celles obtenues avec un nombre illimité de processeurs.

Avec 9 processeurs, notre décomposition est généralement plus rapide que la méthode sans décomposition, puisqu'une fois la décomposition effectuée, le temps nécessaire au reste du processus de résolution tend à être égal ou proche du temps nécessaire à la résolution du plus difficile des sous-problèmes. De plus, sur ces instances, le temps de décomposition est généralement négligeable.

Étant donné que nous tentons ici d'employer un nombre réduit de processeurs, il est préférable de conserver le nombre de sous-problèmes le plus bas possible, et ce sans annihiler les bénéfices de la décomposition. Le nombre moyen de sous-problèmes est de 33,44 sur ces instances, avec un maximum de 142. Notez cependant que seules trois instances dépassaient le nombre de 100 sous-problèmes. Ainsi, le nombre de sous-problèmes à résoudre peut rendre notre nombre de processeurs insuffisant pour certaines instances.

Le coefficient d'accélération obtenu en utilisant m

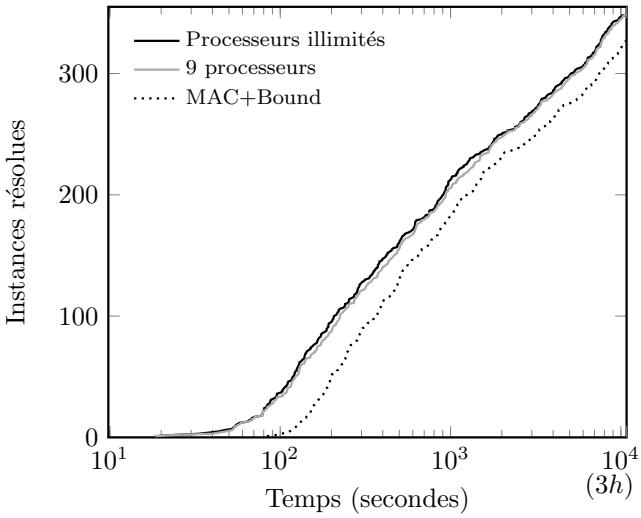

FIGURE 5 – Une comparaison du nombre d'instances résolues pour différentes limites de temps (il ne s'agit pas du cumul des temps) pour la méthode standard et pour la TR-décomposition, avec un nombre illimité hypothétique de processeurs. Une troisième courbe montre que 9 processeurs seraient suffisants pour résoudre autant d'instances dans la limite du temps imparti.

processeurs se trouve être inférieur à m. cette différence s'explique par la présence de nombreuses redondances entre sous-problèmes : différents processeurs font des vérifications similaires, perdant ainsi du temps.

Résoudre les sous-problèmes en parallèle réduit l'impact de ces redondances, mais ce point souligne le fait qu'utiliser la TR-décomposition avec un unique processeur (ou un nombre très faible par rapport au nombre de sous-problèmes générés) à notre disposition donnerait des résultats médiocres lorsque les sous-problèmes ont de grandes intersections, car un même processeur pourrait résoudre les mêmes parties du problème à plusieurs reprises. Pour cette raison, il peut être utile d'ajouter une phase de prétraitement visant à réduire la taille des intersections entre sous-problèmes.

5 Réduction du nombre de sous-problèmes par fusion

Bon nombre de couples de sous-problèmes présentent de grandes intersections, ce qui signifie que nous risquons de résoudre plusieurs fois des parties similaires du problème : le même sous-graphe commun serait construit par deux fils d'exécution sans que nous n'en tirions aucun bénéfice. Afin de limiter ces redondances tout en réduisant le nombre de sous-problèmes, il est possible de *fusionner* certains de ces sous-problèmes. Plus formellement, deux sous-problèmes peuvent être remplacés par un nouveau sous-problème en calculant l'union des valeurs autorisées par les deux sous-problèmes d'origine. Notez que les solutions se trouvant dans les sous-problèmes initiaux sont toujours présentes dans la version fusionnée, car les domaines des variables ne peuvent que grandir durant cette opération.

Dans le but d'empêcher les sous-problèmes de grossir jusqu'à revenir à une taille comparable à celle du problème initial non décomposé, ce qui rendrait la TR-décomposition inutile, il est possible de fixer une limite, en prenant par exemple comme critère la taille de l'espace de recherche du sous-problème formé par la fusion. Dans de telles conditions, la fusion de deux sous-problèmes est interdite si cela créerait un sous-problème dont la taille de l'espace de recherche dépasse la limite choisie. Cette limite peut être déterminée en fonction de la plus grande taille d'espace de recherche trouvée parmi les sous-problèmes. L'idée derrière ce choix est d'empêcher l'apparition de sous-problèmes qui seraient à même de ralentir le processus global de résolution, puisque c'est souvent le sous-problème le plus difficile qui se montre déterminant pour le temps de résolution en parallèle. Il est possible de combiner cette limite avec d'autres. Nous utilisons par exemple une comparaison de la somme des tailles d'espaces de recherche des deux problèmes considérés et de celle du sous-problème qui résulterait d'une fusion de ces deux sous-problèmes. Si la somme calculée est bien plus grande que cette taille qui pourrait être obtenue, cela signifie que l'intersection des espaces de recherche des deux sous-problèmes considérés a une taille non négligeable et qu'il serait intéressant de les fusionner.

Pour résumer ce principe, nous introduisons ici la notion de *gain*, qui exprime l'évolution de la taille totale de l'espace de recherche au cours d'une fusion hypothétique :

Définition 12 *Soient S_1 et S_2 deux sous-problèmes distincts. Le* gain *offert par la fusion de S_1 et de S_2 est donné par le rapport suivant : $gain(S_1, S_2) = \frac{size(S_1)+size(S_2)}{size(S_1 \cup S_2)}$, où $size(S_i)$ correspond à la taille de l'espace de recherche de S_i, et $S_1 \cup S_2$ représente le sous-problème qui serait créé si S_1 et S_2 venaient à être fusionnés.*

Il s'ensuit logiquement que deux sous-problèmes offrant un gain strictement supérieur à 1, une fois fusionnés, permettent à la complexité théorique d'être plus basse qu'en résolvant ces sous-problèmes séparément. À l'inverse, un gain strictement inférieur à 1 signifie que fusionner ces sous-problèmes augmenterait

la complexité théorique.

Bien entendu, fusionner des sous-problèmes lorsque leur nombre a déjà été ramené au nombre de processeurs disponibles ou à un nombre encore plus faible ne présente a priori aucun avantage. Des fusions supplémentaires risqueraient de créer des sous-problèmes extrêmement difficiles, car les limites utilisées, basées sur les tailles d'espaces de recherche, ne sont pas infaillibles et sont avant tout des moyens d'estimations de la difficulté. Nos expériences nous ont montré que ce risque vaut certes la peine d'être pris lorsque le nombre de sous-problème est grand mais qu'il valait mieux stopper le processus de fusion dès que ce nombre devenait convenablement faible.

6 Expériences avec fusions de sous-problèmes

Ces nouvelles expériences emploient la fusion de sous-problèmes afin d'effectuer un prétraitement. Le but principal de ces fusions est de réduire l'impact des intersections entre sous-problèmes : fusionner deux sous-problèmes permet d'éviter que le programme ne fasse de mêmes vérifications plusieurs fois. Cependant, fusionner de manière irréfléchie peut ralentir la résolution dans son ensemble en créant des sous-problèmes exceptionnellement longs à résoudre.

Au cours de ces expériences, les restrictions suivantes ont été imposées lors du prétraitement :

— Le gain minimal qu'une paire de sous-problèmes doit offrir pour prétendre à la fusion est de 1, excepté dans les cas où une autre valeur est explicitement indiquée.

— Toute fusion qui créerait un sous-problème ayant un espace de recherche plus grand que le plus grand observable parmi les sous-problèmes courants est interdite, comme expliqué en section 5.

— Nous n'effectuons qu'une fusion à la fois, en choisissant à chaque cycle la paire de sous-problèmes offrant le meilleur gain, tout en ignorant celles qui ne respectent pas la condition précédemment exposée.

Les résultats obtenus sont visibles dans les figures 6 et 7. Dans la limite de temps de 3 heures, plusieurs instances sont uniquement résolues par les méthodes usant de la TR-décomposition.

En ce qui concerne le nombre de sous-problèmes, nous observons une moyenne de 33,44 (resp. 31,69, 20,63 et 8,22) sur les 1 177 instances examinées si aucune fusion n'est effectuée (resp. si les sous-problèmes sont fusionnés avec un gain minimal de 1, 0,5 et 0). Cette moyenne n'a pas pu descendre en dessous de 8, même avec un gain minimal de 0 (auquel cas la seule limite fixée est celle relative à la taille de l'espace de recherche le plus grand parmi les sous-problèmes), car huit processeurs étaient utilisés et les fusions sont nécessairement stoppées dès lors qu'il ne reste pas plus d'un sous-problème par processeur. Il est également intéressant de noter que le nombre de sous-problèmes générés durant ces expériences correspondait généralement à moins de un pourcent du maximum théorique (qui est égal au nombre de nœuds dans le graphe de compatibilité).

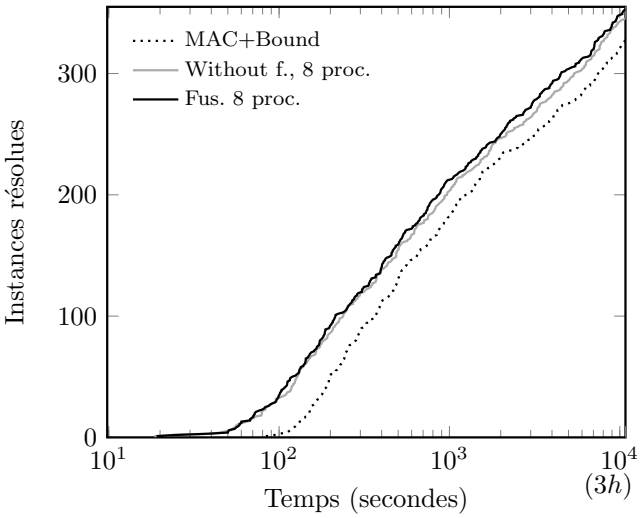

FIGURE 6 – Ce graphique montre notre *baseline* (un unique CSP résolu avec MAC et une borne inférieure) ainsi que deux autres courbes pour la TR-décomposition sur 8 processeurs : l'une avec fusions (avec un gain minimal de 1) et l'autre sans fusion.

Concernant le temps nécessaire pour réaliser les fusions, il n'a pas dépassé le tiers de seconde pour un gain minimal de 1 ou les deux tiers de seconde pour 0.

Le tableau 2 montre l'impact du choix de gain minimal de fusion sur les performances. Des fusions excessives ont un effet négatif sur certaines instances, tandis que ne pas fusionner du tout fait également baisser les performances.

La figure 6 montre qu'utiliser un gain minimal de fusion de 1 permet effectivement à la TR-décomposition d'être plus efficace. La tableau 2 résume ces résultats de manière plus lisible.

On constate que le choix d'un gain minimal de 1 offre de meilleurs résultats que les autres possibilités ici passées en revue.

Puisque la fusion de sous-problèmes est avant tout conçue pour diminuer les redondances entre sous-problèmes, cette méthode peut rapprocher de n le coefficient d'accélération offert par l'usage de n processeurs. Notez cependant que, malgré ces redondances, 25 instances ont été résolues plus rapidement avec dé-

Tableau 2 – Le nombre d'instances résolues après différents laps de temps, pour différentes politiques de fusion : g0 n'a pour seule limite que la règle interdisant de créer des sous-problèmes plus grands que le plus grand courant ; g0,5 ignore les paires de sous-problèmes offrant un gain inférieur à 0,5 ; g1 est notre méthode déjà présentée dans les figures précédentes ; « S. f. » n'effectue aucune fusion. « M+B » correspond aux résultats obtenus avec MAC+Bound.

T (s)	M+B	F. g0	F. g0,5	F. g1	S. f.
0	0	0	0	0	0
100	2	21	29	34	32
200	50	81	84	92	86
400	111	132	135	140	138
800	163	187	190	196	185
1 600	220	233	233	238	231
3 200	249	269	270	278	268
6 400	289	306	309	313	310
10 800	327	347	351	353	345

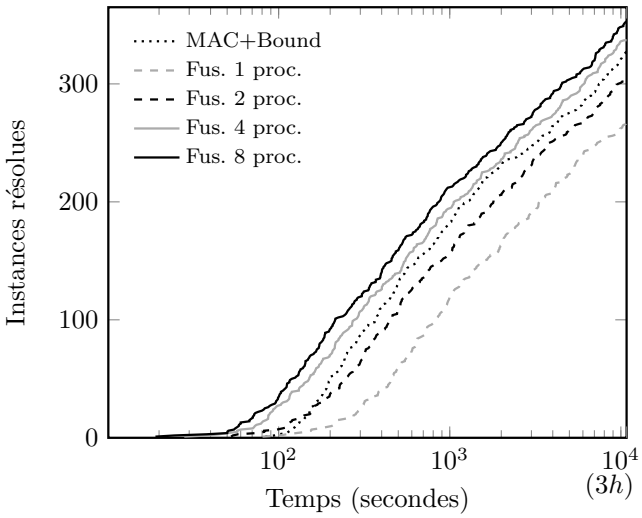

Figure 7 – Ce graphique montre l'évolution des performances pour différents nombres de processeurs. Les fusions sont ici toujours activées (gain minimal de 1). La *baseline*, présente dans les figures précédentes, est également affichée.

composition même en utilisant un unique processeur, parfois plus de deux fois plus rapidement.

7 Conclusion

Nous présentons dans cet article une adaptation de la TR-décomposition, préalablement définie par [10], au problème du plus grand sous-graphe commun induit. Cette méthode s'appuie sur une triangulation du graphe de compatibilité du problème et permet de décomposer ce problème en sous-problèmes indépendants, pouvant être résolus en parallèle. Nous présentons une technique de fusion visant à réduire les redondances inhérentes à cette approche. Les expériences montrent que la TR-décomposition est une méthode adaptée au problème du MCIS sur les instances difficiles. Sur ces instances, le temps nécessaire à la décomposition est généralement rentabilisé grâce à un raccourcissement significatif de la phase de résolution proprement dite. De plus, fusionner des sous-problèmes de manière avisée ouvre de nouvelles perspectives et améliorent ces résultats.

À l'avenir, plusieurs points feront l'objet d'une attention plus particulière.

D'autres méthodes de décomposition présentent un certain attrait, comme effectuer plusieurs triangulations pour équilibrer les sous-problèmes. Là encore, il nous sera nécessaire de trouver un bon compromis entre le temps que nécessite la décomposition appliquée et sa qualité.

Il sera également possible d'adapter la TR-décomposition à l'approche consistant à recherche une clique maximale dans le graphe de compatibilité (intégral ou, avec des sous-problèmes, divisé) plutôt que de résoudre des CSP.

Il serait intéressant d'employer des instances plus réalistes, provenant de cas scientifiques réels ou tout du moins présentant des structures plus caractéristiques.

Références

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

[2] H. G. Barrow and R. M. Burstall. Subgraph isomorphism, matching relational structures and maximal cliques. *Information Processing Letters*, 4(4) :83–84, 1976.

[3] Assef Chmeiss, Philippe Jégou, and Lamia Keddar. On a generalization of triangulated graphs for domains decomposition of csps. In *IJCAI*, pages 203–208. Citeseer, 2003.

[4] Donatello Conte, Pasquale Foggia, and Mario Vento. Challenging complexity of maximum common subgraph detection algorithms : A performance analysis of three algorithms on a wide database of graphs. *J. Graph Algorithms Appl.*, 11(1) :99–143, 2007.

[5] Adnan Darwiche. Recursive conditioning. *Artificial Intelligence*, 126(1) :5–41, 2001.

[6] Rina Dechter and Robert Mateescu. And/or search spaces for graphical models. *Artificial intelligence*, 171(2) :73–106, 2007.

[7] Matjaz Depolli, Janez Konc, Kati Rozman, Roman Trobec, and Dusanka Janezic. Exact parallel maximum clique algorithm for general and protein graphs. *Journal of chemical information and modeling*, 53(9) :2217–2228, 2013.

[8] 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) :1–16, 1999.

[9] Delbert R Fulkerson, Oliver A Gross, et al. Incidence matrices and interval graphs. *Pacific J. Math*, 15(3) :835–855, 1965.

[10] Philippe Jégou. Decomposition of domains based on the micro-structure of finite constraint-satisfaction problems. In *AAAI*, volume 93, pages 731–736, 1993.

[11] Philippe Jégou and Cyril Terrioux. Hybrid backtracking bounded by tree-decomposition of constraint networks. *Artificial Intelligence*, 146(1) :43–75, 2003.

[12] Uffe Kjaerulff. Triangulation of graphs - algorithms giving small total state space. Technical report, Judex R.R. Aalborg., Denmark, 1990.

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

[14] Ciaran McCreesh and Patrick Prosser. Multi-threading a state-of-the-art maximum clique algorithm. *Algorithms*, 6(4) :618–635, 2013.

[15] Ciaran McCreesh and Patrick Prosser. The shape of the search tree for the maximum clique problem, and the implications for parallel branch and bound. *arXiv preprint arXiv :1401.5921*, 2014.

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

[17] Maël Minot and Samba Ndojh Ndiaye. Searching for a maximum common induced subgraph by decomposing the compatibility graph. In *Bridging the Gap Between Theory and Practice in Constraint Solvers, CP2014-Workshop*, pages 1–17, September 2014.

[18] Samba Ndojh Ndiaye and Christine Solnon. Cp models for maximum common subgraph problems. In *Principles and Practice of Constraint Programming–CP 2011*, pages 637–644. Springer, 2011.

[19] Thierry Petit, Jean-Charles Régin, and Christian Bessière. Specific filtering algorithms for over-constrained problems. In *Principles and Practice of Constraint Programming – CP 2001*, pages 451–463. Springer, 2001.

[20] J W Raymond, E J Gardiner, and P Willett. Rascal : calculation of graph similarity using maximum common edge subgraphs. *The Computer Journal*, 45(6) :631–644, 2002.

[21] Jean-Charles Régin, Mohamed Rezgui, and Arnaud Malapert. Embarrassingly parallel search. In *Principles and Practice of Constraint Programming - 19th International Conference, CP 2013, Uppsala, Sweden, September 16-20, 2013. Proceedings*, pages 596–610, 2013.

[22] Jean-Charles Régin, Mohamed Rezgui, and Arnaud Malapert. Improvement of the embarrassingly parallel search for data centers. In *Principles and Practice of Constraint Programming - 20th International Conference, CP 2014, Lyon, France, September 8-12, 2014. Proceedings*, pages 622–635, 2014.

[23] Daniel Sabin and Eugene C Freuder. Contradicting conventional wisdom in constraint satisfaction. In *Principles and Practice of Constraint Programming*, pages 10–20. Springer, 1994.

[24] Christine Solnon. Solving maximum clique problems with ant colony optimization liris.cnrs.fr/csolnon/antclique.html, 2006.

[25] Christine Solnon and Serge Fenet. A study of ACO capabilities for solving the maximum clique problem. *J. Heuristics*, 12(3) :155–180, 2006.

[26] Philippe Vismara. Programmation par contraintes pour les problemes de plus grand sous-graphe commun. In *JFPC'11 : Journées Francophones de Programmation par Contraintes*, pages 327–335, 2011.