Techniques de Décomposition pour l'Isomorphisme de Sous-Graphe

Stéphane Zampelli, Martin Mann, Yves Deville, Rolf Backofen

Actes JFPC 2008

Techniques de Décomposition pour l'Isomorphisme de Sous-Graphe

Stéphane Zampelli[1], Martin Mann[2], Yves Deville[1], Rolf Backofen[2]

(1) Université catholique de Louvain,
Department of Computing Science and Engineering,
2, Place Sainte-Barbe 1348 Louvain-la-Neuve (Belgium)
(2) Albert-Ludwigs-University Freiburg,
{stephane.zampelli,yves.deville}@uclouvain.be
Chair for Bioinformatics, Institute of Computer Science,
Georges-Koehler-Allee 106, D-79110 Freiburg (Germany)
{mmann,backofen}@informatik.uni-freiburg.de

Résumé

Les techniques existantes de décomposition sont inefficaces sur des problèmes dont le graphe de contraintes initial est complet. Nous nous intéressons en particulier au problème de l'isomorphisme de sous-graphe, et montrons comment utiliser une approche hybride de décomposition statique et dynamique pour ce problème. L'idée sous-jacente est de précalculer une heuristique statique sur un sous-ensemble du réseau de contraintes, de suivre cette heuristique statique, et d'utiliser pour le reste de la recherche une propagation forte ainsi que une détection dynamique de décomposition du réseau de contraintes. Les résultats expérimentaux montrent que pour des graphes à faibles degrés, notre méthode de décomposition résout plus d'instances que les algorithmes dédiés pour l'isomorphisme de sous-graphe et pour les approches par contraintes existantes.

1 Introduction

La recherche de motif dans des graphes est une application centrale dans beaucoup de domaines [1] et peut être modélisée en utilisant la programmation par contraintes [11, 15, 18]. Dans cet article, nous montrons comment appliquer les techniques de décomposition pour le problème de l'isomorphisme de sous-graphe (SIP).

Un problème de satisfaction de contraintes (CSP) peut être associé avec un réseau de contraintes, qui représente les contraintes actives ainsi que les variables. Pendant la recherche, le réseau de contraintes perd de la structure lorsque des variables sont instanciées et que des contraintes qui seront toujours vérifiées sont enlevées. Le réseau de contraintes peut être composé de deux réseaux formant des composantes indépendantes, induisant des calculs redondants dû à la combinaison des instanciations des composantes indépendantes.

Ce calcul redondant peut être évité en utilisant une technique de *décomposition* qui comprend deux étapes. La première étape détecte les composantes indépendantes du CSP, en examinant le réseau de contraintes sous-jacent. La seconde étape exploite ces composantes indépendantes en résolvant les CSP indépendants partiels correspondants, et combine leur solution. La décomposition peut se produire à n'importe quel noeud de l'arbre de recherche, i.e. au noeud racine ou bien dynamiquement durant la recherche. En programmation par contraintes, les techniques de décomposition ont été étudiées au moyen du concept de recherche ET/OU [13]. La recherche ET/OU détecte la décomposition d'un CSP en introduisant un arbre de recherche qui possède des noeuds ET qui sont une extension des noeuds OU des arbres de recherche classiques. La taille minimale d'un arbre de recherche ET/OU est exponentielle dans la largeur de l'arbre alors que la taille de l'arbre minimal OU est exponentielle dans la largeur

de chemin, et n'est jamais pire que la taille de l'arbre de recherche OU.

La détection de décomposition s'effectue habituellement de deux manières. La première analyse le réseau de contraintes initial, en créant une heuristique appelée *pseudo-tree*. Cette dernière structure encode l'heuristique statique à suivre durant la recherche ainsi que le moment où le réseau de contraintes est décomposable [5]. La seconde manière est d'analyser dynamiquement le réseau de contraintes, c'est-à-dire à chaque noeud de l'arbre de recherche [3]. Une telle approche dynamique est intéressante si une propagation forte (e.g. l'arc consistance) est présente, mais elle induit un coût calculatoire supplémentaire.

Ces techniques de décomposition sont spécifiques. Sans le calcul d'une heuristique appropriée, la décomposition peut se produire très tardivement dans l'arbre de recherche, si bien que le temps de calcul supplémentaire pour la détection de la décomposition induit un algorithme inefficace. Néanmoins, il existe des algorithmes dédiés calculant ces heuristiques, sous la forme de noeuds séparateurs de graphes ou d'arrêtes enlevant tous les cycles [9, 13, 14].

Cependant, ces algorithmes ne parviennent pas à calculer de bonnes heuristiques sur des CSP contenant des contraintes globales, parce que ces CSPs ont un graphe de contraintes initialement complet. En effet, de tels algorithmes présupposent un graphe de contraintes peu dense. Dans le SIP, le graphe de contraintes initial est complet étant donné la présence d'une contrainte global *alldiff*. Cette contrainte empêche les algorithmes cités plus haut d'être appliqués. Un autre problème de l'analyse statique est l'impossibilité de prévoir la décomposition du réseau de contraintes obtenu par le retrait de contraintes inutiles suite à la propagation. Pour exploiter la réduction du réseau de contraintes, une analyse dynamique de ce réseau durant la recherche est nécessaire. L'analyse dynamique est très importante pour la résolution SAT [12] et CSP [3]. Malheureusement, une analyse dynamique nécessite un temps de calcul supplémentaire qui ralentit la recherche.

Dans cet article nous montrons comment dépasser ces limitations en combinant des approches de décomposition statique et dynamique pour utiliser les techniques de décomposition pour le SIP. L'idée sous-jacente est de suivre l'heuristique calculée statiquement jusqu'à ce qu'une première décomposition se produise (ou a de grandes chances de se produire), et de passer ensuite à un niveau de propagation élevé. Comme nous le montrerons dans la partie expérimentale, cette idée est un point important pour une application efficace d'une recherche utilisant la décomposition (comme les techniques ET/OU) pour le SIP.

Pour résoudre le problème du SIP, beaucoup d'approches ont été proposées, depuis des méthodes générales à des algorithmes spécifiques à certaines classes de graphes. L'approche considérée comme l'état de l'art est l'algorithme dédié appelé *VF*, disponible gratuitement avec le code source C++ de la librairie appelée `vflib` [2]. En programmation par contraintes, certains auteurs [11, 15] ont montré que l'appariement de graphes peut être formulé comme un CSP, et ont défendu l'idée que la programmation par contraintes pourrait être une approche efficace. Notre modélisation [18] est basé sur ces travaux. Dans [18], nous avons montré que une approche CSP est compétitive avec des algorithmes dédiés sur un jeu de test représentant des graphes avec des topologies variées. En ce qui concerne la décomposition, Valiente et al. [17] ont montré comment utiliser les techniques de décomposition en vue de résoudre plus efficacement l'homomorphisme de graphe. [17] montre que, si le graphe motif initial est constitué de plusieurs composants déconnectés, alors apparier chaque composante séparément est équivalent à les apparier toutes ensembles. Notre article se distingue sur trois points de ce travail. Tout d'abord, nous considérons spécifiquement le SIP au lieu du problème plus général de l'homomorphisme de sous-graphe. Ce dernier problème est plus facile en ce qui concerne les techniques de décomposition car le graphe de contraintes est constitué seulement des arcs du graphe motif initial. De plus, nous appliquons la décomposition dynamiquement alors que [17] décompose seulement statiquement le graphe motif initial. Enfin, pour l'isomophisme de sous graphe, [17] signale que l'algorithme proposé ne peut pas décomposer le graphe motif, mais uniquement le graphe cible.

Objectifs et résultats

Dans cet article nous étudions les limites de l'application directe de l'état de l'art des techniques de décomposition (aussi bien statiques que dynamiques) pour des problèmes avec des contraintes globales ; nous montrons qu'une application directe est inefficace pour le SIP. Nous développons une approche hybride et concevons une heuristique de recherche pour obtenir des décompositions. Nous montrons que l'approche CP utilisant la technique de décomposition proposée surpasse les algorithmes de l'état de l'art, et résout plus d'instances sur certaines classes de problèmes (instances peu denses avec beaucoup de solutions). L'article est structuré comme suit. La Section 2 introduit une méthode de décomposition capable de détecter la décomposition à n'importe quelle étape de la recherche. La Section 3 applique cette méthode de décomposition au SIP et la spécialise. Les résultats expérimentaux étudiant l'efficacité de notre approche

sont présentés dans la Section 4. La Section 5 conclut l'article.

2 Décomposition

Dans cette section nous montrons comment définir et détecter la décomposition durant la recherche. Les Sections 2.1 et 2.2 définissent une méthode de décomposition capable de détecter la décomposition à n'importe quelle étape durant la recherche, en considérant que nous ne connaissons pas *a priori* le moment où la décomposition va se produire. La section 2.3 montre que notre méthode est capable de calculer les mêmes décompositions que la recherche ET/OU [13], dans laquelle la recherche est précalculée sur la représentation du graphe de contraintes, et les événements de décomposition sont connus à l'avance. La méthode de recherche ET/OU s'est montrée être efficace pour une large classe de problèmes de réseau de contraintes. Comme nous le montrerons dans la Section 3, notre méthode est efficace pour le SIP alors que la méthode ET/OU n'est pas applicable parce que les décompositions ne peuvent pas être précalculées.

2.1 Définitions préliminaires

Un *problème de satisfaction de contraintes (CSP)* P est un triplet $(X, \mathcal{D}, C)$ où $X = \{x_1, \ldots, x_n\}$ est un ensemble de variables et $\mathcal{D} = \{D_1, \ldots, D_n\}$ est un ensemble de domaines (i.e. un ensemble fini de valeurs), chaque variable x_i est associée avec un domaine D_i, et C est un ensemble fini de contraintes avec $scope(c) \subseteq X$ pour tout $c \in C$, où $scope(c)$ est l'ensemble des variables comprises dans la contrainte c. Une contrainte c sur un ensemble de variables définit une relation entre les variables. Une *solution* du CSP est une affectation de chaque variable dans X à une valeur dans son domaine associé de telle sorte qu'aucune contrainte $c \in C$ soit violée. Nous notons $Sol(P)$ l'ensemble des solutions d'un CSP P.

Un *CSP partiel* $\hat{P}$ d'un CSP $P \equiv (X, \mathcal{D}, C)$ est un CSP $(\hat{X}, \hat{\mathcal{D}}, \hat{C})$ où $\hat{X} \subseteq X$, $\forall \hat{D}_k \in \hat{\mathcal{D}} : \hat{D}_k \subseteq D_k$ et $\hat{C} \subseteq C$. Notons que puisque $\hat{P}$ est un CSP, nous avons $scope(\hat{c}) \subseteq \hat{X}$ pour tout $\hat{c} \in \hat{C}$.

2.2 Décomposition de CSPs et de graphes

Cette sous-section définit la notion de décomposition d'un CSP. Un CSP est décomposable en CSPs partiels si le CSP et sa décomposition ont les mêmes solutions.

Définition 1. *Un CSP P est* décomposable *en CSPs partiels $P_1, \ldots, P_n$ ssi :*

- $\forall\ s \in Sol(P) : \exists\ s_1, \ldots, s_k \in Sol(P_1), \ldots, Sol(P_k) : s = \cup_{i \in [1,k]} s_i$
- $\forall\ s_1, \ldots, s_k \in Sol(P_1), \ldots, Sol(P_k) : \exists\ s \in Sol(P) : s = \cup_{i \in [1,k]} s_i$.

Cette définition générale de la notion de décomposition peut être instanciée à deux cas pratiques. La première définition correspond à l'intuition directe d'une décomposition : un CSP est décomposable s'il peut être séparé en CSPs partiels disjoints. Cette décomposition est appelée 0-décomposition puisque aucune variable n'est partagée entre les CSPs partiels.

Définition 2. *Un CSP $P = (X, \mathcal{D}, C)$ est* 0-décomposable *en CSPs partiels $P_1, \ldots, P_n$ avec $P_i = (X_i, \mathcal{D}_i, C_i)$ ssi $\forall\ 1 \leq i < j \leq n : X_i \cap X_j = \emptyset$, $\cup_{i \in [1,k]} X_i = X$, $\cup_{i \in [1,k]} \mathcal{D}_i = \mathcal{D}$, $\cup_{i \in [1,k]} C_i = C$.*

La seconde définition trouve plus de décompositions en autorisant les CSPs partiels à avoir des variables instanciées en commun. Cette décomposition est appelée 1-décomposition puisque les variables communes entre les CSPs partiels ont un domaine de taille 1.

Définition 3. *Un CSP $P = (X, \mathcal{D}, C)$ est* 1-décomposable *en CSPs partiels $P_1, \ldots, P_k$ avec $P_i = (X_i, \mathcal{D}_i, C_i)$ ssi $\forall\ 1 \leq i < j \leq n : x \in (X_i \cap X_j) \Rightarrow |D_x| = 1$, $\cup_{i \in [1,k]} X_i = X$, $\cup_{i \in [1,k]} \mathcal{D}_i = \mathcal{D}$, $\cup_{i \in [1,k]} C_i = C$.*

Le lien avec la définition générale est direct. Si un CSP P est *0-décomposable* ou *1-décomposable* en CSPs partiels $P_1, \ldots, P_k$, alors P est décomposable en CSPs partiels $P_1, \ldots, P_k$. Des définitions 2 et 3, il s'ensuit que :

Propriété 1. *Si un CSP $P = (X, \mathcal{D}, C)$ est 0-décomposable en $P_1, \ldots, P_k$, alors P est 1-décomposable en $P_1, \ldots, P_k$. De plus P peut être 1-décomposable en $P'_1, \ldots, P'_{k'}$ avec $k' \geq k$ en recouvrant partiellement les problèmes P'_i.*

Le calcul redondant pendant la résolution du CSP est effectuée à chaque fois que un CSP est 0- ou 1-décomposable en k CSPs partiels $P_1, \ldots, P_k$. Par exemple, si les solutions de P_1 sont calculées en premier, alors pour chaque solution de P_1 toutes les solutions de $P_2, \ldots, P_k$ sont calculées. Dès lors, $P_2, \ldots, P_k$ sont résolus $|Sol(P_1)|$ fois et ces redondances peuvent être exponentielles dans la taille du CSP. Ceci peut être évité en résolvant les problèmes partiels indépendamment. La détection de la décomposition du CSP en CSPs partiels indépendants peut être effectuée en utilisant le concept de graphe de contraintes.

Un *graphe* $G = (V, E)$ peut être représenté par un *ensemble de noeuds* V et un *ensemble d'arrêtes* $E \subseteq V \times V$, où une arrête (u, v) est une paire de

noeuds. Les noeuds u et v sont les noeuds de l'arrête (u, v). Nous considérons des graphes dirigés et non dirigés. Un *sous-graphe* d'un graphe $G = (V, E)$ est un graphe $G' = (V', E')$ avec $V' \subseteq V$ et $E' \subseteq E$ de telle sorte que $\forall_{(u,v)\in E'} : u, v \in V'$. Un graphe G est dit être *simplement connecté* si et seulement si il existe au plus un chemin simple entre n'importe quelle paire de noeuds dans G.

Définition 4. *Le* graphe de contraintes *d'un CSP (partiel) $P = (X, \mathcal{D}, C)$ est un graphe non dirigé $G^P = (V, E)$ où $V = X$ et $E = \{(x_i, x_j) \mid \exists\ c \in C : x_i, x_j \in scope(c)\}$.*

Notons que toutes les variables comprises dans une contrainte globale forment une clique dans G^P. Ce graphe de contraintes est aussi appelé *primal graph* [4]. Il existe une manière syntaxique standard pour décomposer un CSP, basé sur son graphe de contraintes.

Définition 5. *Un graphe $G = (V, E)$ est* décomposable *en k sous-graphes $G_1, \ldots, G_k$ ssi $\forall_{1\leq i<j\leq k}$: $V_i \cap V_j = \emptyset$, $\cup_{i\in[1,k]} V_i = V$, et $\cup_{i\in[1,k]} E_i = E$.*

La Propriété 2 montre que l'on doit calculer des composantes disjointes du graphe de contraintes pour détecter des CSPs indépendants. Cela peut être fait en temps linéaire.

Propriété 2. *Etant donné un CSP $P = (X, \mathcal{D}, C)$ avec son graphe de contraintes G, pour tout $k \geq 1$, le graphe de contraintes G de P est décomposable dans $G_1, \ldots, G_k$, ssi P est 0-décomposable en $P_1, \ldots, P_k$ ssi P est 1-décomposable en $P'_1, \ldots, P'_m$ avec $m \geq k$.*

Preuve - Le premier ssi est direct. Pour le second, nous pouvons construire une 1-décomposition $P_1, \ldots, P_m$ de P d'une décomposition $G_1, \ldots, G_k$ de G, avec $m \geq k$. La construction est décrite par $k = 1$ (i.e. $P_1 = P$), et peut être facilement généralisée. Soit $G = (V, E)$ le graphe de contraintes de P. Soit $V_s = \{x \in V \mid |D_x| = 1\}$. Transformer G en G' où G' est le graphe G sans les variables qui ont un domaine singleton. Plus formellement, $G' = (V', E')$ avec $V' = V \backslash V_s$ et $E' = (V' \times V') \cap E$. Supposons G' est décomposable en $G'_1, \ldots, G'_m$ $(m \geq 1)$. Alors, les noeuds associés à des variables qui ont un domaine singleton, et leurs arrêtes associées sont ajoutées à G'_i, donnant $G^1_i = (V^1_i, E^1_i)$. Plus formellement $G^1_i = (V^1_i, E^1_i)$ où $V^1_i = V'_i \cup V_s$ et $E^1_i = (V^1_i \times V^1_i) \cap E$. Les graphes $G^1_1, \ldots, G^1_m$ sont les graphes de contraintes des CSPs partiels P_i provenant de la 1-décomposition de P. ■

La propriété ci-dessus est spécialement utile quand $k = 1$. Dans ce cas, la 0-décomposition ne décompose pas le CSP, alors que la 1-décomposition peut le décomposer.

2.3 Lien avec l'arbre de recherche ET/OU

Une autre approche pour définir des CSPs décomposables est d'utiliser le concept d'arbre de recherche ET/OU défini au moyen de pseudo-trees [13].

Définition 6. *Etant donné un graphe non dirigé $G = (V, E)$, un arbre enraciné et dirigé $T = (V, E')$ définit sur tous ses noeuds est appelé* pseudo-tree *de G si n'importe quel arc de E qui n'est pas inclus dans E' est un arc de retour, càd qu'il connecte un noeud avec son ancêtre dans T.*

Définition 7. *Etant donné un CSP $P = (X, \mathcal{D}, C)$, son graphe de contraintes G^P et un pseudo-tree T^P de G^P, l'arbre de recherche associé ET/OU possède des niveaux successifs de noeuds OU et de noeuds ET. Les noeuds OU sont labélisés x_i et correspondent aux variables. Les noeuds ET sont labélisés $< x_i, v_k >$ et correspondent à l'affectation de valeurs v_k dans les domaines des variables. La racine de l'arbre de recherche ET/OU est un noeud OU, labélisé avec la racine du pseudo-tree T^P. Les enfants d'un noeud OU x_i sont des noeuds ET labélisés avec des affectations $< x_i, v_k >$, consistants le long du chemin à partir de la racine. Les enfants d'un noeud ET $< x_i, v_k >$ sont des noeuds OU labélisés avec les enfants de la variable x_i dans T^P.*

Les noeuds OU représentent des solutions alternatives, alors que les noeuds ET représentent des décompositions en problèmes partiels indépendants, qu'il faut tous résoudre. Lorsque le pseudo-tree est une chaîne, l'arbre de recherche ET/OU coïncide avec l'arbre de recherche habituel OU.

En suivant l'ordre induit par un pseudo-tree T^P du graphe de contraintes du CSP P, la notion de 1-décomposition coïncide avec la décomposition induite par l'arbre de recherche ET/OU.

Propriété 3. *Etant donné un CSP $P = (X, \mathcal{D}, C)$, un pseudo-tree T^P défini sur un graphe de contraintes de P et un chemin p de longueur l $(l \geq 1)$ à partir du noeud racine de T^P à un noeud ET p_l, le CSP P où toutes les variables dans le chemin p sont assignées est 1-décomposable en $P_1, \ldots, P_k$ où k est le nombre de noeuds OU successeurs dans T^P du noeud final p_l.*

Preuve - Soit $y_1, \ldots, y_k$ $(k \geq 1)$ être les noeuds successeurs OU du noeud final p_l dans T^P. Nous notons $tree(y_i)$ l'arbre enraciné à y_i dans T^P. Soit $X_s = \{v \in X | v \in p\}$. Alors construisons les CSPs partiels $P_i = (X_i, \mathcal{D}_i, C_i)$ $(i \in [1, k])$:

$$\begin{aligned} X_i &= X_s \cup \{v \in X \mid v \in tree(y_i)\} \\ \mathcal{D}_i &= \{D_x \in \mathcal{D} \mid x \in X_i\} \\ C_i &= \{c \in C \mid scope(c) \subseteq X_i\}. \end{aligned}$$

Il est clair que $\cup_{i\in[1,k]} C_i = C$ puisqu'il n'existe pas de contraintes entre deux différents $tree(y_i)$ dans T^P, par définition d'un pseudo-tree. ■

Comme l'explique la section suivante, ni l'approche statique ni l'approche dynamique des arbres de recherches ET/OU est efficace pour le SIP. Dans ce problème, le graphe de contraintes est complet, et donc le pseudo-tree est une chaîne, si bien que l'arbre de recherche ET/OU est équivalent à un arbre de recherche OU. Cependant le CSP P devient 1-décomposable pendant la recherche et une approche dynamique est nécessaire pour détecter les décompositions. Malheureusement, cette détection est très coûteuse en calcul, comme démontré dans la Section 4.

3 Appliquer la décomposition au SIP

3.1 Définition du problème de l'isomorphisme de sous-graphe

Un *problème d'isomorphisme de sous-graphe* entre un graphe motif $G_p = (V_p, E_p)$ et un graphe cible $G_t = (V_t, E_t)$ consiste à décider s'il existe un sous-graphe de G_t auquel G_p est isomorphe. Plus précisément, le but est de trouver une fonction injective $f : V_p \to V_t$ telle que $\forall (u, v) \in E_p : (f(u), f(v)) \in E_t$. Ce problème NP-complet est aussi appelé le problème du monomorphisme de graphe dans la littérature. La fonction f est appelée *fonction d'appariement*. Nous supposons que les graphes sont dirigés. Les graphes non dirigés sont un cas particulier où les arrêtes sont remplacés par deux arcs dirigés en sens opposés.

Le modèle CSP $P = (X, \mathcal{D}, C)$ de l'isomorphisme de sous-graphe représente une fonction totale $f : V_p \to V_t$. Cette fonction totale peut être modélisée avec $X = x_1, \ldots, x_n$ où x_i est une variable entière correspondant au i$^{\text{ème}}$ noeud de G_p et $D(x_i) = V_t$. La condition injective est modélisée avec la contrainte globale $\texttt{alldiff}(x_1, \ldots, x_n)$. La condition d'isomorphisme est traduite dans un ensemble de n contraintes d'arité k $MC_i \equiv (x_i, x_j) \in E_t$ pour tout $x_i \in V_p$ et pour tout x_j voisin de x_i. Etant donné le modèle ci-dessus, le graphe de contrainte du CSP, appelé le graphe de contraintes SIP, est le graphe $G^P = (V^P, E^P)$ où $V^P = X$ et $E^P = E_p \cup E_{\neq}$. Notons que E_p représente tous les propagateurs des contraintes binaires MC_i tandis que $E_{\neq}$ décrit la décomposition en contraintes de la contrainte globale `alldiff`, i.e. une clique ($E_{\neq} = V_p \times V_p$). De plus, le CSP SIP contient une contrainte globale alldiff, qui empêche toute décomposition en utilisant une recherche statique ET/OU. L'implémentation, la comparaison avec des algorithmes dédiés, et l'extension de l'isomorphisme de sous-graphe aux variables graphe et aux variables fonction sont décrites dans [18].

3.2 Décomposition du SIP

Cette sous-section explique comment décomposer le problème du SIP. Nous montrons tout d'abord pourquoi une recherche ET/OU pose problème en étudiant le graphe de contraintes du SIP.

Recherche ET/OU statique : A cause de la contrainte `alldiff`, le graphe de contraintes du SIP correspond au graphe complet $K_{|V_p|}$. Le pseudo-tree calculé sur le graphe de contraintes de toute instance SIP est une chaîne, et ne détecte aucune décomposition. De plus, le graphe de contraintes initial du SIP n'est pas 1-décomposable. Une analyse statique du CSP SIP ne renvoie aucune décomposition.

Obtenir des décompositions semble difficile. Cependant, puisque les variables sont instanciées durant la recherche, une 1-décomposition peut survenir à certains noeuds de l'arbre de recherche. Une détection dynamique de la 1-décomposition nous donne une première manière de détecter des décompositions pour le SIP.

Recherche ET/OU dynamique : Une analyse dynamique du graphe de contraintes SIP, prend en compte la possible disparition de certaines contraintes (parce qu'elles sont vraies pour le reste de la recherche) et du résultat de la propagation. Cette détection dynamique est donc utile pour les CSPs où la propagation est forte. Le désavantage réside dans les calculs supplémentaires à cause de la consistance élevée et de la détection dynamique de la décomposition.

Notre notion de 1-décomposition enlève des variables instanciées durant le processus de décomposition. Enlever les contraintes vraies pour le reste de la recherche peut être fait facilement pour la contrainte `alldiff`, en enlevant une arrête $(x_i, x_j) \in E^P$ représentant $x_i \neq x_j$ quand $D_i \cap D_j = \emptyset$ $(i \neq j)$. Nous redéfinissons le graphe de contraintes du SIP comme un graphe de contraintes pour les contraintes de morphisme avec un graphe de contraintes dynamique pour la contrainte `alldiff`.

Définition 8. *Etant donné le CSP $P = (X, \mathcal{D}, C)$ d'une instance du SIP, son* graphe de contraintes SIP *est le graphe non dirigé $G = (V, E^{MC} \cup E^{\neq})$, où $V = X$, $E^{MC} = \{(x_i, x_j) \in E_p \mid x_i, x_j \in X\}$ et $E^{\neq} = \{(i, j) \in X \times X \mid D_i \cap D_j \neq \emptyset\}$.*

Etant donné la structure particulière du graphe de contraintes SIP, il est possible de spécialiser et de simplifier la détection d'une 1-décomposition.

Propriété 4. *Soit $P = (X, \mathcal{D}, C)$ un modèle CSP d'une instance SIP, et soit $G = (V, E^{MC} \cup E^{\neq})$ son graphe de contraintes SIP. Soit $M = (V', E')$ le graphe de contraintes sans variables assignées, i.e. avec $V' = \{x \in X \mid |D_x| > 1\}$ et $E' = (V' \times V') \cap E^{MC}$.*

Alors P est 1-décomposable en $P_1, \ldots, P_m$ ssi M est décomposable en $M_1, \ldots, M_m$ et $D(M_i) \cap D(M_j) = \emptyset$ $(1 \leq i < j \leq m)$ avec $D(M_i)$ l'union des domaines des variables associées aux noeuds de M_i.

La propriété ci-dessus explique que la décomposition de M est une condition nécessaire. Nous pouvons dès lors concevoir une heuristique de décomposition de M, qui peut provoquer une décomposition de P.

Une approche directe consiste à détecter la 1-décomposition à chaque noeud de l'arbre de recherche. Quand le CSP devient 1-décomposable en CSPs partiels, ceux-ci sont calculés séparément en noeuds ET. Comme le démontre la section expérimentale, cette stratégie se montre beaucoup plus lente qu'un arbre de recherche standard OU, pour deux raisons :

1. La décomposition est testée à chaque noeud de l'arbre de recherche. Commencer une telle détection du noeud racine de l'arbre de recherche est inutile, et beaucoup de calculs superflus sont effectués.
2. Il n'y a pas de garantie qu'une décomposition puisse se produire.

Sur base de cette observation, nous présentons une approche hybride combinant le meilleur de la stratégie statique et dynamique de recherche.

L'Approche Hybride : Comme expliqué plus haut, même une approche dynamique ET/OU, qui teste la décomposition sur le graphe de contraintes dynamique, n'est pas assez efficace pour être concurrentiel avec les algorithmes pour le SIP. Nous suggérons une approche hybride pour régler cette difficulté. L'idée est la suivante :

1. calculer une heuristique de décomposition statique sur le graphe de contraintes réduit
2. appliquer une recherche avec un niveau de propagation *forward checking* qui utilise l'heuristique statique de décomposition jusqu'à la première décomposition ou jusqu'à ce que un nombre fixé de variables soit assigné.
3. passer ensuite à une détection dynamique avec une propagation arc-consistante.

Cette méthode assure que l'approche dynamique coûteuse est utilisée seulement quand une décomposition est présente, ou du moins probable après que la propagation forte ait été activée. Jusqu'à ce moment, une approche de détection avec une propagation faible est utilisée pour alléger le graphe de contraintes.

Nous présentons maintenant deux heuristiques dédiées que nous avons appliquées pour calculer l'heuristique statique de décomposition.

3.3 Heuristiques

Nous présentons à présent deux heuristiques basées sur la Propriété 4 qui visent à réduire le nombre de tests de décomposition, et à favoriser la décomposition. L'idée générale est de détecter un sous-ensemble de variables déconnectant le graphe de contraintes du morphisme en composantes disjointes, puisqu'il s'agit d'une condition nécessaire pour la 1-décomposition. Le processus de recherche assignera d'abord ces variables. Le test de la 1-décomposition est effectué lorsque toutes ces variables sont instanciées. Le test est aussi effectué sur tous les autres noeuds de la recherche.

3.3.1 L'heuristique de cycle (h1)

L'objectif de l'heuristique de cycle est de trouver un ensemble de noeuds S dans le graphe de morphisme $CG^{MC} = (X, E^{MC})$ (voir Def. 8) tel que ce graphe privé de ces noeuds est simplement connecté. Lorsque les variables associées à S sont affectées, toute affectation ultérieure décomposera le graphe de morphisme. Trouver l'ensemble minimal de noeuds est connu sous le nom de *minimal cycle cutset* et est un problème NP-difficile. Nous proposons ici une approximation linéaire pour trouver les noeuds des cycles du graphe. L'Algorithme 1 a une complexité temporelle de $O(|V_p|)$. L'efficacité de cet algorithme sur différentes classes de problèmes est démontré dans la section expérimentale. Son avantage principal est sa simplicité.

entrée: $G = (X, E)$ le CG^{MC}
sortie : Les noeuds des cycles de G

$All \leftarrow X$
$T \leftarrow \emptyset$
while $(\exists\, n \in X \mid Degree(n) == 1)$ **do**
  $T \leftarrow T \cup \{n\}$
  enlever le noeud n de G
end
renvoyer $All \setminus T$

Algorithm 1: Sélection des variables *body*.

3.3.2 Partitionnement de graphe (h2)

Le partitionnement de graphe est une technique bien connue qui vise à résoudre des problèmes de graphe difficiles par une approche *divide and conquer*. Il peut également être utilisé pour séparer le graphe de contraintes de morphisme en deux graphes de taille égale.

Définition 9. *Etant donné un graphe $G = (V, E)$, un k-partitionnement du graphe G est une partition de V en k sous-ensembles, $V_1, \ldots, V_k$, tels que $V_i \cap V_j = \emptyset$ pour $i \neq j$, $\cup_i V_i = V$, telle que $|V_1| = \cdots = |V_k|$*

et telle que le nombre d'arrêtes de E dont les noeuds incidents appartiennent à différents sous-ensembles est minimal (appelé le edgecut*).*

A partir du edgecut de graphe de contraintes de morphisme, nous pouvons facilement déduire un sous-ensemble de variables.

Définition 10. *Etant donné un 2-partitionnement de G, un* nodecut *est un ensemble de noeuds contenant un noeud de chaque arrête dans le edgecut.*

Trouver un edgecut minimum est un problème NP-difficile pour $|V_i| \geq 3$, mais peut être résolu en temps polynomial pour $|V_i| = 2$ par appariement (voir [7], page 209). Nous utilisons une approximation locale rapide [10].

4 Expérimentations

4.1 Objectifs

L'objectif de nos expériences est d'identifier la classe de graphes où la décomposition est efficace. Nous comparons notre méthode de décomposition sur différentes classes d'instance du SIP avec des modèles CP standards et *vflib*, l'algorithme standard de référence pour le SIP [2]. Nous comparons également notre méthode de décomposition avec la décomposition standard. Les différentes heuristiques présentées en Section 3.5 sont également testées.

4.2 Instances

Les instances proviennent du jeu de test de `vflib` décrit dans [6]. Il y a plusieurs classes de graphes, les graphes aléatoires, des graphes à degrés bornés, et des graphes `mesh`. Les graphes cibles ont une taille de n, et la taille relative du graph motif est notée α. Pour les graphes aléatoires, le graphe cible a un nombre fixe de noeuds n et il existe un arc dirigé entre deux noeuds avec une probabilité η. Le graphe motif est généré avec la même probabilité η, mais son nombre de noeud est αn. Si le graphe généré n'est pas connecté, des arcs supplémentaires sont ajoutés jusqu'à ce que le graphe soit connecté. Pour les graphes aléatoires, n prend sa valeur dans $[20, 40, 80, 100, 200, 400, 800, 1000]$, η dans $[0.01, 0.05, 0.1]$, et α dans $[20\%, 40\%, 60\%]$. Il y a donc 69 classes de graphes aléatoires. Dans une classe d'instance dénotée `si2-r001-m200`, on a $\alpha = 20\%$, $\eta = 0.01$, et $n = 200$ noeuds.

Les graphes mesh k-connectés sont des graphes où chaque noeud est connecté avec k noeuds voisins. Les graphes mesh irréguliers k-connectés sont constitués de graphes mesh k-réguliers où l'on ajoute aléatoirement des arcs de manière uniforme. Le nombre d'arcs ajoutées est ρn. Pour les graphes mesh k-connectés, n prend ses valeurs dans $[16, \ldots, 1096]$, k dans $[2, 3, 4]$, et ρ dans $[0.2, 0.4, 0.6]$. Pour un graphe mesh irrégulier k-connecté, dans une classe d'instances `si2-m4Dr6-m625`, on a $\alpha = 20\%$, $k = 4$, $\rho = 0.6$ et $n = 625$ noeuds.

Cent graphes sont générés pour chaque classe d'instances. Pour des graphes aléatoires, nous avons également générés 100 instances additionnelles où le graphe cible a 1600 noeuds, pour chaque valeur possible de η et α. Nous avons utilisé le générateur disponible avec le jeu de test [6].

4.3 Modèles

Plusieurs modèles ont été considérés. Tout d'abord, nous utilisons l'implémentation officielle de `vflib`. Ensuite les modèles CP classiques sont utilisés, appelés `CPFC` et `CPAC`. Le modèle `CPFC` est un modèle où toutes les contraintes utilisent le forward checking et l'heuristique de variable sélectionne la première variable qui est incluse dans le nombre maximum de de contraintes (appelée `maxcstr`), et utilise la variable dont le domaine est le plus petit en cas d'égalité. Le modèle `CPAC` est le même excepté qu'il utilise une version arc-consistante de la contrainte `MC`, en plus de n arc-consitantes contraintes `alldiff`sur les voisins de chaque noeud du graphe motif, comme proposé dans [16]. Des résultats similaires sont observés avec une seule contrainte globale `alldiff`qui porte sur tous les noeuds, mais avec des performances légèrement plus faibles.

Le modèle `CP+Dec` attend que 30% des variables soient instanciées en suivant une heuristique de variable, appelée `minsize`, qui sélectionne la variable non instanciée avec le plus petit domaine. Il teste ensuite à chaque noeud de l'arbre de recherche si une décomposition s'est produite en utilisant une heuristique de variable `maxcstr`. Le modèle `CP+Dec+h1` utilise l'heuristique de cycle ; dès que les noeuds appartenant aux cycles du graphe motif sont instanciés en utilisant une heuristique de variable `minsize` (jusqu'à 30% des noeuds du graphe motif), la décomposition est testée à chaque noeud de l'arbre de recherche et suit une heuristique de variable `maxcstr`. Le modèle `CP+Dec+h2` utilise une heuristique de partitionnement de graphe ; dès que les variables appartenant au *nodecut* sont instanciées (jusqu'à 30% des noeuds du graphe motif), la décomposition est testée à chaque noeud de l'arbre de recherche et suit une heuristique de variable `maxcstr`.

Toutes les expériences ont été effectuées sur une grappe de 16 machines (AMD Opteron(tm) 875 2.2Ghz avec 2Gb of RAM). Toutes les exécutions sont limitées à 10 minutes. Pour chaque exécution, nous cherchons toutes les solutions.

4.4 Description des tables

La Table 1 montre les résultats pour les graphes aléatoires et la Table 2 pour les graphes mesh irréguliers. Chaque ligne décrit l'exécution de 100 instances pour une classe particulière. La colonne N indique le nombre moyen de solutions parmi les instances résolues. La colonne % indique le nombre d'instances qui ont été résolues endéans 10 minutes. La colonne μ indique que le temps moyen sur les instances résolues et la colonne σ indique la déviation standard correspondante. La colonne D indique le nombre d'instances qui ont utilisé la décomposition parmi les instances résolues. Le colonne $\#D$ indique le nombre moyen de décomposition qui se sont produites parmi les instances résolues. La colonne S indique la taille moyenne de l'ensemble de variables initial calculé par l'heuristique `h1` or `h2`. La Table 3 donne le degré moyen et la variance pour les différentes classes d'instances. Pour chaque classe d'instances de la Table 1 et 2, les résultats des meilleurs algorithmes sont en gras.

4.5 Analyse

Nous commençons l'analyse par considérer les graphe aléatoires (Table 1). Nous comparons tout d'abord `vflib` avec les modèles `CPFC` et `CPAC`. Pour toutes les instances `si2-*` et `si6-*` , le modèle `CPAC` est le meilleur en temps moyen et en % d'instances résolues excepté pour la classe `si2-r001-m200`, où `CPFC` est la meilleure.

Nous comparons à présent les méthodes de décomposition pour les graphes aléatoires (deuxième table dans la Table 1).

Tout d'abord, nous nous concentrons sur les classes `si2-r001-*`. Les modèles `CP+Dec+h1` et `CP+Dec+h2` trouvent de meilleures décompositions que le modèle `CP+Dec`. Même si `CP+Dec` induit plus de décompositions, le nombre d'instance utilisant la décomposition (voir colonne D) est plus grande pour `CP+Dec+h1` et `CP+Dec+h2` que pour `CP+Dec`. Ceci montre le coût calculatoire supplémentaire d'une approche de décomposition dynamique habituelle. Cependant, le nombre d'instances utilisant la décomposition tend vers zéro pour les instances `m1600`. Ceci est dû au fait que les graphes ont des degrés plus grands quand leur taille augmente (voir Table 3). Ceci peut être observé en regardant la colonne S : la taille du sous-ensemble initial de variables à instancier devient proche de 100% lorsque la taille augmente. En effet, notre méthode de décomposition est moins efficace que le modèle `CPAC` pour `si2-r001-m800` et `m1600`.

Nous nous concentrons à présent sur les classes `si6-r01-*`. Comme mentionné plus tôt, ces instances ont des graphes denses. L'ensemble initial de variables à instancier est l'ensemble complet des noeuds du graphe motif pour `CP+Dec+h1` et `CP+Dec+h2`. Aucune décomposition ne se produit. Pourquoi dès lors les modèles `CP+Dec+h*` sont plus efficaces que toutes les autres méthodes pour ces classes ? Parce que dans la classe `si6-r01-*`, l'approche `CP+Dec+h1` revient à un niveau de consistance hybride qui se trouve entre `CPFC` et `CPAC` avec une heuristique de variable `minsize` pendant la phase de forward checking.

Pour les graphes aléatoires, la méthode de décomposition avec heuristique est particulièrement utile pour les graphes peu denses avec beaucoup de solutions, alors qu'un modèle hybride qui commence avec du forward checking et qui passe ensuite de l'arc-consistance semble être le meilleur choix pour des graphes plus denses et où il y a peu de solutions. L'algorithme `vflib` est clairement moins efficace sur toutes ces classes d'instances. Les expériences sur d'autres classes de graphes aléatoires, que nous ne montrons pas ici faute de place, confirment cette analyse.

Nous analysons maintenant les graphes mesh irréguliers. Nous observons Table 3 que le degré moyen des classes `si2-m4Dr6-*` sont plus élevées que pour les classes `si6-m4Dr6-*`. Nous comparons tout d'abord `vflib` et les modèles CP sans décomposition. Pour les classes peu denses `si2-m4Dr6-*`, `CPFC` est le meilleur modèle, alors que pour les classes plus denses `si6-m4Dr6-*`, `vflib` est le meilleur. Nous n'avons pas d'explication particulière pour ce comportement, et ceci reste une question ouverte. En ce qui concerne les méthodes de décomposition, les mêmes remarques que pour les graphes aléatoires s'appliquent. Le modèle `CP+Dec` produit moins de décompositions que les modèles `CP+Dec+h*`. De plus, les modèles `CP+Dec+h*` sont les meilleurs modèles pour les instances peu denses avec beaucoup de solutions. Lorsque le degré moyen des instances augmentent (voir Table 3) et le nombre de solutions diminue, les méthodes de décomposition deviennent moins efficaces. En effet, pour `si6-m4Dr6-m1296`, la meilleur méthode est `vflib`, mais notre approche par décomposition résout également toutes les instances et aide l'approche CP à diminuer son temps moyen.

Résumé L'application des méthodes standards de décomposition `CP+Dec` mène à des performances inférieures que l'application directe des modèles standard CP (`CPFC`, `CPAC`) et `vflib`. Sur la plupart des classes, l'heuristique de cycle (`h1`) est meilleur que l'heuristique de partitionnement (`h2`). Sur les graphes peu denses avec beaucoup de solutions, et sur les graphes mesh irréguliers, notre méthode de décomposition a de meilleurs performances que l'approche CP standard et que `vflib`. Pour des graphes denses, les modèles hybrides CP entre `CPAC` et `CPFC` avec une

TAB. 1 – Graphes aléatoires, toutes les solutions sont calculées.

Bench		vflib			CPAC			CPFC		
	N	%	μ	σ	%	μ	σ	%	μ	σ
si2-r001-m200	61E+6	72	74	115	83	56	109	85	41	76
si2-r001-m400	17E+8	2	248	118	10	106	156	7	288	177
si2-r001-m800	28E+7	0	-	-	**14**	**166**	**133**	1	153	-
si2-r001-m1600	2500	16	203	202	**81**	**224**	**93**	0	-	-
si6-r01-m200	1	100	2	3	100	9	11	100	12	17
si6-r01-m400	1	66	99	133	89	156	116	50	190	137
si6-r01-m800	1	7	235	153	0	-	-	5	389	125
si6-r01-m1600	1	0	-	-	0	-	-	39	499	51

Bench		CP+Dec					CP+Dec+h1						CP+Dec+h2					
	N	%	μ	σ	D	#D	%	μ	σ	D	#D	S	%	μ	σ	D	#D	S
si2-r001-m200	61E+6	94	49	100	91	9244	**98**	**6**	**40**	98	1834	0.2	87	23	48	71	909	0.2
si2-r001-m400	17E+8	15	160	177	15	35655	**75**	**68**	**125**	75	2268	0.4	29	212	218	22	196	0.3
si2-r001-m800	28E+7	0	-	-	0	12	4	227	254	4	21	0.6	12	256	239	8	0	0.6
si2-r001-m1600	2500	0	-	-	0	0	7	165	199	1	0	0.8	0	-	-	0	0	0.9
si6-r01-m200	1	94	148	153	0	0	**100**	**0**	**0**	0	0	1	100	0	0	0	0	1
si6-r01-m400	1	2	179	220	0	0	**100**	**2**	**1**	0	0	1	100	4	6	0	0	1
si6-r01-m800	1	0	-	-	0	0	**100**	**46**	**35**	0	0	1	100	46	39	0	0	1
si6-r01-m1600	1	0	-	-	0	0	**74**	**479**	**71**	0	0	1	54	435	79	0	0	1

TAB. 2 – Mesh irréguliers, toutes les solutions sont calculées.

Bench		vflib			CPAC			CPFC		
	N	%	μ	σ	%	μ	σ	%	μ	σ
si2-m4Dr6-m625	88E+5	89	23	50	94	21	38	95	6	27
si2-m4Dr6-m1296	17E+7	16	135	137	33	178	123	38	107	154
si6-m4Dr6-m625	3.31	100	7	43	100	29	4	100	9	4
si6-m4Dr6-m1296	10.38	**100**	**13**	**55**	100	233	30	100	113	65

Bench		CP+Dec					CP+Dec+h1						CP+Dec+h2					
	N	%	μ	σ	D	#D	%	μ	σ	D	#D	S	%	μ	σ	D	#D	S
si2-m4Dr6-m625	88E+5	35	223	151	35	0.7	**100**	**6**	**22**	96	5.4	0.5	94	6	21	88	5.5	0.3
si2-m4Dr6-m1296	17E+7	3	120	36	3	0.1	**63**	**67**	**109**	63	4	0.5	49	163	170	49	3.9	0.5
si6-m4Dr6-m625	3.3	8	105	32	0	0	**100**	**7**	**3**	6	0.1	0.8	100	22	26	6	0.1	0.7
si6-m4Dr6-m1296	10.3	0	-	-	0	0	100	65	20	41	0.6	0.7	77	223	161	29	0.4	0.7

TAB. 3 – Degré moyen pour l'ensemble des classes utilisées.

Bench	degré	
	μ	σ
si2-r001-m200	2.30	0.14
si2-r001-m400	2.89	0.14
si2-r001-m800	3.99	0.18
si2-r001-m1600	6.80	0.19
si6-r01-m200	3.29	0.14
si6-r01-m400	5.27	0.16
si6-r01-m800	9.76	0.15
si6-r01-m1600	19.20	0.17
si2-m4Dr6-m625	3.51	0.26
si2-m4Dr6-m1296	3.53	0.20
si6-m4Dr6-m625	5.12	0.16
si6-m4Dr6-m1296	5.19	0.14

heuristique `minsize` est le meilleur choix. Pour des graphes mesh irréguliers denses, `vflib`, le modèle standard CP et notre méthode de décomposition résolvent toutes les instances, bien que `vflib` soit plus efficace.

5 Conclusion

Notre question initiale était d'étudier l'application des techniques de décomposition telle que la recherche ET/OU pour des problèmes avec des contraintes globales, en particulier pour le SIP. Nous avons montré que c'est en effet possible en utilisant une approche hybride de technique statique et dynamique et une analyse dédiée de la structure du problème. Pour le SIP, des décompositions peuvent être obtenues en utilisant une heuristique statique et un niveau de propagation faible (forward checking). Dès que le problème devient (potentiellement) décomposable, le processus de recherche passe à une recherche qui teste dynamiquement la recherche et qui utilise un niveau de propagation élevé (arc consistance). Nous avons montré que notre approche de décomposition hybride est capable de battre l'algorithme de référence `vflib` pour des graphes peu denses avec beaucoup de solutions. Comme travaux futurs, nous aimerions étudier l'utilisation de notre méthode de décomposition pour des problèmes de *motif discovery* où la résolution du SIP est utilisée comme un outil d'énumération [8].

Remerciements Cette recherche est supportée par la Région Wallonne (projet Transmaze, WIST516207), et par le PAI Moves (Politique scientifique belge).

Références

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

[2] Luigi Pietro Cordella, Pasquale Foggia, Carlo Sansone, and Mario Vento. An improved algorithm for matching large graphs. In *3rd IAPR-TC15 Workshop on Graph-based Representations in Pattern Recognition*, pages 149–159. Cuen, 2001.

[3] R. Dechter and R. Mateescu. AND/OR search spaces for graphical models. *Artificial Intelligence*, 171(2-3) :73–106, 2007.

[4] Rina Dechter. *Constraint Processing (The Morgan Kaufmann Series in Artificial Intelligence)*. Morgan Kaufmann, May 2003.

[5] Rina Dechter and Robert Mateescu. The impact of AND/OR search spaces on constraint satisfaction and counting. In *Proc. of the CP'2004*, 2004.

[6] P. Foggia, C. Sansone, and M. Vento. A database of graphs for isomorphism and sub-graph isomorphism benchmarking. *CoRR*, cs.PL/0105015, 2001.

[7] Michael R. Garey and David S. Johnson. *Computers and Intractability ; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1990.

[8] Joshua A. Grochow and Manolis Kellis. Network motif discovery using subgraph enumeration and symmetry-breaking. In *RECOMB*, pages 92–106, 2007.

[9] G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J. Sci. Comput.*, 20 :359–392, 1998.

[10] George Karypis and Vipin Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J. Sci. Comput.*, 20(1) :359–392, 1998.

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

[12] Wei Li and Peter van Beek. Guiding real-world SAT solving with dynamic hypergraph separator decomposition. In *Proc. of the 16th IEEE International Conference on Tools with Artificial Intelligence*, 2004.

[13] R. Mateescu. *AND/OR Search Spaces for Graphical Models*. PhD thesis, 2007.

[14] Robert Mateescu and Rina Dechter. AND/OR cutset conditioning. In *Proc. of the IJCAI'2005*, 2005.

[15] Michael Rudolf. Utilizing constraint satisfaction techniques for efficient graph pattern matching. In *Theory and Application of Graph Transformations*, number 1764 in Lecture Notes in Computer Science, pages 238–252. Springer, 1998.

[16] Jean-Charles Régin. Developpement d'outils algorithmiques pour l'intelligence artificielle : application a la chimie organique. *PhD Thesis*, 1995.

[17] Gabriel Valiente and Conrado Martínez. An algorithm for graph pattern-matching. In *Proc. 4th South American Workshop on String Processing*, volume 8 of *International Informatics Series*, pages 180–197. Carleton University Press, 1997.

[18] Stéphane Zampelli, Yves Deville, and Pierre Dupont. Approximate constrained subgraph matching. In *Principles and Pratice of Constraint Programming*, volume 3709 of *Lecture Notes in Computer Science*, pages 832–836, 2005.