Décomposition arborescente et cohérence locale souple dans les CSP pondérés

Simon de Givry[1], **Thomas Schiex**[1] **et Gérard Verfaillie**[2]

[1] INRA & [2]ONERA - DCSD, Toulouse, France

{degivry,tschiex}@toulouse.inra.fr, verfaillie@cert.fr

Résumé

Plusieurs approches récentes pour résoudre les modèles graphiques (réseaux Bayésiens avec contraintes) exploitent simultanément une décomposition du graphe et le maintien d'une propriété de cohérence locale. La décomposition de graphe exploite la structure du problème, offrant des bornes sur la complexité spatiale et temporelle, tandis que la propagation des contraintes dures conduit en pratique à des améliorations en terme de mémoire et de temps à l'intérieur de ces bornes de complexité théorique.

Parallèlement, l'extension des propriétés de cohérence locale au cas des réseaux de contraintes pondérées a permis d'importants gains d'efficacité pour les méthodes fondées sur le principe de séparation et évaluation. En fait, ces cohérences locales souples établissent de manière incrémentale des minorants permettant de couper dans l'arbre de recherche et de guider les heuristiques.

Dans cet article, nous considérons la combinaison d'une méthode exploitant la décomposition arborescente avec le maintien d'une propriété de cohérence locale souple pour résoudre des problèmes de satisfaction de contraintes pondérées. L'interaction complexe entre ces deux mécanismes amène à considérer différentes approches ayant différentes propriétés théoriques. Il apparaît que la combinaison la plus prometteuse sacrifie un peu les bornes théoriques pour une meilleure efficacité en pratique.

1 Introduction

Le traitement des modèles graphiques est un problème important en Intelligence Artificielle. Ces dernières années, pour résoudre les problèmes de satisfaction, d'optimisation et de comptage, plusieurs algorithmes ont été proposés qui exploitent simultanément une décomposition arborescente du graphe du problème et la propagation d'informations dures en utilisant les techniques de renforcement de cohérence locale. Cela inclut en particulier les algorithmes tels que Recursive Conditioning (RC) [6], Backtrack bounded by Tree Decomposition (BTD) [28], AND-OR Tree and Graph Search [22, 21], tous en relation avec Pseudo-Tree Search [13].

Implicitement ou non, tous ces algorithmes exploitent les propriétés des décompositions arborescentes [2] qui capturent des informations d'indépendance structurelle, dans le but d'obtenir des bornes de complexité en temps et en espace pour ces algorithmes. La combinaison avec la propagation des informations dures augmente les capacités d'élagage dans l'arbre de recherche, conduisant à de meilleures bornes de complexité en pratique en temps et en espace à l'intérieur des bornes de complexité originales. Notons cependant que dans le contexte d'une recherche arborescente, ces bornes sont obtenues au prix d'une restriction sur l'ordre d'affectation des variables (voir [1] sur ce sujet).

Dans ce papier, nous nous intéressons plus particulièrement aux problèmes d'optimisation exprimés dans le cadre des réseaux de contraintes valuées ou pondérées [25]. Les réseaux de contraintes pondérées offrent un cadre très général pour modéliser de nombreuses applications dans des domaines variés tels que l'*allocation de ressources* [3], la *vente aux enchères* [23] et la *bio-informatique* [26, 8].

Dans une première partie, nous montrons que, avec un affaiblissement limité des bornes de complexité théorique existantes, une combinaison particulière de décomposition arborescente et de recherche par séparation et évaluation peut être définie. Dans une seconde partie, nous exploitons l'extension des cohérences locales au cas des réseaux de contraintes pondérées [24]. Cette extension a permis d'améliorer l'efficacité des recherches par séparation et évaluation en utilisant des propriétés de cohérences locales de plus en plus fortes [4, 20, 19, 11]. Elles permettent de maintenir incrémentalement un minorant de

l'optimum du sous problème courant et aussi de contribuer à des ordres de choix de variables et de valeurs mieux informés. Introduites dans une approche exploitant une décomposition arborescente, elles devraient permettre une amélioration de la complexité pratique en temps et en espace et aussi une exploration arborescente mieux guidée.

Prises ensemble, ces deux parties impliquent un traitement complexe des pondérations, conduisant à différentes alternatives possibles, ayant chacune différentes propriétés théoriques et différente efficacité en pratique.

2 Cadre et notations

Un CSP binaire pondéré (WCSP) est un triplet (X, D, W). $X = \{1, \ldots, n\}$ est un ensemble de n variables. Chaque variable $i \in X$ a un domaine fini $D_i \in D$ de valeurs qui peuvent lui être affecté. La taille du plus grand domaine est notée d. W est un ensemble de contraintes souples, appelées aussi fonctions de coûts. Une contrainte binaire souple $W_{ij} \in W$ est une fonction $W_{ij} : D_i \times D_j \mapsto [0, k]$ où k est un coût entier maximum donné correspondant à une affectation complètement interdite exprimant ainsi une contrainte dure. Si elles n'existent pas, nous ajoutons dans W des fonctions de coûts unaires pour toutes les variables telles que $W_i : D_i \mapsto [0, k]$ et aussi une contrainte d'arité nulle $W_\varnothing$ qui correspond à un coût constant que doit payer n'importe quelle affectation. Toutes ces fonctions de coûts additionnelles retournent comme valeur initiale zéro, laissant ainsi la sémantique du problème inchangée.

Le problème est alors de trouver une affectation complète de l'ensemble des variables de coût minimum : $\min_{(a_1,a_2,\ldots,a_n)\in\prod_i D_i}\{W_\varnothing + \sum_{i=1}^{n} W_i(a_i) + \sum_{W_{ij}\in W} W_{ij}(a_i, a_j)\}$, ce qui correspond à un problème d'optimisation NP-dur.

Le graphe des contraintes du WCSP est noté $G = (X, E)$ avec un sommet pour chaque variable et une arête (i, j) pour chaque contrainte binaire $W_{ij} \in W$. Une décomposition arborescente de ce graphe est définie par un arbre noté (C, T). L'ensemble des noeuds de l'arbre est noté $C = \{C_1, \ldots, C_m\}$ où C_e correspond à un ensemble de variables ($C_e \subset X$) appelé regroupement ou cluster. T est l'ensemble des arêtes reliant les clusters et formant un arbre (un graphe acyclique connecté). L'ensemble des clusters C doit couvrir toutes les variables ($\bigcup_{C_e\in C} C_e = X$) et toutes les contraintes ($\forall (i, j) \in E, \exists C_e \in C$ *s.t.* $i, j \in C_e$). De plus, si une variable i apparaît dans deux clusters C_e et C_g, alors i doit aussi apparaître dans tous les clusters C_f sur l'unique chemin allant de C_e à C_g dans (C, T).

L'intérêt pratique de cette définition s'appuie sur le fait que tous les problèmes acycliques peuvent être résolus avec de bonnes bornes de complexité théorique. Ainsi, une décomposition arborescente découpe un problème en sous-problèmes (clusters) organisés sous forme d'un graphe acyclique, avec pour variables reliant deux clusters adjacents

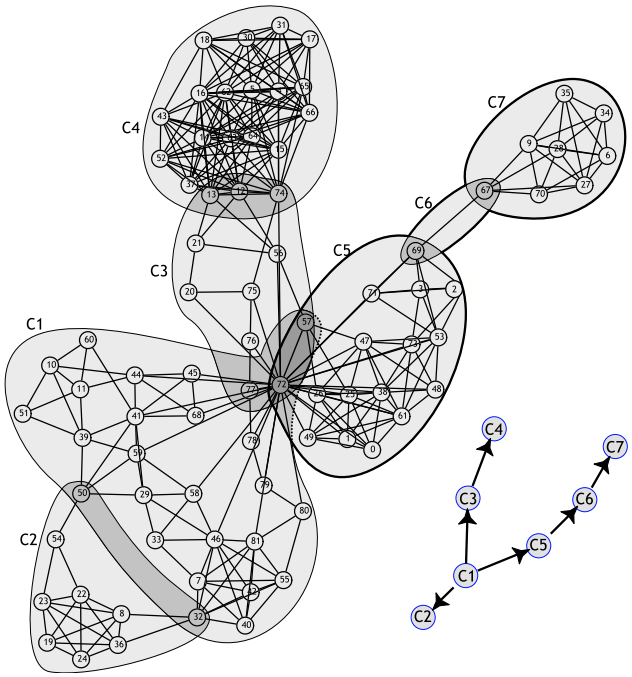

Figure 1: Le graphe de contraintes de l'instance SCEN-06 (prétraitée par l'élimination des variables de degré inférieur ou égal à deux) du problème RLFAP recouvert par sept clusters, ainsi que l'arbre de décomposition associé, enraciné au cluster C1. Le sous-problème P_5 est mis en évidence, avec un séparateur $S_5 = \{57, 72\}$. La contrainte $W_{57,72}$ dans S_5 ne fait pas partie de P_5. Notons également le séparateur $S_6 = \{69\}$ de P_6.

(variables dont le retrait déconnecte les sous-problèmes) celles obtenues en faisant l'intersection des deux clusters.

Ceci est illustré par la Figure 1 où le graphe d'un problème d'allocation de fréquences est recouvert par des clusters définissant une décomposition arborescente. Cette figure montre qu'il existe bien en dehors des problèmes purement aléatoires des problèmes réels fortement structurés dont la décomposition arborescente s'avère exploitable. Ceci est particulièrement vrai dans le cas des problèmes de conception où la modularité est importante ainsi que dans de nombreux problèmes où la localité (spatiale ou temporelle) des données implique de telles structures.

La *largeur d'arbre* d'une décomposition arborescente donnée (C, T) est égale à $\max_{C_e\in C}\{|C_e|\} - 1$, notée w par la suite. La largeur d'arbre w^* d'un graphe G est la largeur d'arbre minimum parmi toutes les décompositions arborescentes de G (aussi appelée *largeur induite* de G). Si une racine $C_r \in C$ est choisie, le nombre maximum de variables apparaissant dans un chemin commençant par C_r est appelé la hauteur d'arbre de la décomposition. Trouver une décomposition ayant une largeur d'arbre minimum ou une hauteur d'arbre minimum est un problème NP-dur.

3 Exploitation d'une décomposition arborescente

Pour un WCSP donné, nous considérons une décomposition arborescente (C, T) enraciné arbitrairement au cluster C_1. Nous notons *Father*(C_e) (resp. *Sons*(C_e)) le parent (resp. l'ensemble des fils) de C_e dans T. Le séparateur de C_e est l'ensemble $S_e = C_e \cap$ *Father*(C_e). L'ensemble des variables propres de C_e est noté $V_e = C_e \setminus S_e$. Remarquons que les ensembles V_e définissent une partition de X. Pour chaque variable $i \in X$, nous dénotons par $[i]$ l'index du cluster tel que $i \in V_{[i]}$.

La propriété essentielle d'une décomposition arborescente est que l'affectation de S_e sépare le problème initial en deux sous-problèmes qui peuvent alors être résolus indépendamment. Le premier sous-problème, noté P_e, est défini par toutes les variables de C_e et toutes celles de ses clusters descendants dans T et par toutes les fonctions de coûts impliquant au moins une variable propre de ces clusters. Les contraintes restantes, ainsi que les variables sur lesquelles elles portent, définissent le sous-problème restant.

Cette propriété a été exploitée dans de nombreux algorithmes. Combinée au principe de recherche par séparation et évaluation, cette propriété peut être exploitée en restreignant l'ordre de choix des variables. Supposons que toutes les variables d'un cluster C_e soient affectées avant toutes celles restant dans ses clusters fils et posons A comme étant l'affectation courante. Alors, pour tout cluster $C_f \in$ *Sons*(C_e), et pour l'affectation courante A_f du séparateur S_f, le sous-problème P_f sous l'affectation A_f (noté P_f/A_f) peut être résolu indépendamment du reste du problème. Si la mémoire le permet, le *coût optimal* de P_f/A_f peut être mémorisé, ce qui signifie que le sous-problème ne sera jamais résolu une seconde fois pour la même affectation de S_f.

Si d est la taille maximum des domaines, h la hauteur d'arbre de la décomposition et w la largeur d'arbre, cette idée appliquée récursivement sans mémorisation garantie qu'un cluster C_e ne sera jamais visité plus de fois que d à la puissance du nombre de variables dans le chemin allant de la racine à C_e (ses variables propres exclues). Cela donne une garantie sur le nombre de noeuds visités qui est dominé par $O(d^h)$. Cela concerne Pseudo-Tree Search et AND-OR Tree Search, qui ont été appliqués à la résolution des WCSP dans [18, 22].

Avec mémorisation, un cluster C_e ne sera jamais revisité plus que le nombre d'affectations possibles de son séparateur S_e. Connaissant le nombre de variables dans V_e, cela signifie que le nombre de noeuds est dominé par $O(d^{w+1})$, $w + 1$ étant la taille du plus grand cluster. Ceci donne des algorithmes tels que RC avec mémorisation complète, BTD et AND-OR Graph Search, qui peuvent être considérés comme des algorithmes d'apprentissage structurel. La contrepartie est que la mémoire nécessaire peut être exponentielle dans la taille du plus grand séparateur.

3.1 Exploitation de majorants améliorés

De façon traditionnelle, la recherche par séparation et évaluation en profondeur d'abord est un algorithme de recherche arborescente qui exploite deux bornes. Dans le cas des WCSP, où l'on cherche une solution de coût minimum, la meilleure solution trouvée jusqu'alors procure un majorant de l'optimum. Durant la recherche, à un noeud donné de l'arbre de recherche, un mécanisme dédié permet de calculer un minorant du coût de n'importe quelle affectation complète sous ce noeud courant. Typiquement, lorsque la recherche commence, le majorant est soit égal au coût maximum k soit au coût de la meilleure solution trouvée par exemple par une recherche locale.

Dans le cas d'une combinaison directe d'une décomposition arborescente avec une recherche par séparation et évaluation, telle qu'effectuée dans BTD appliqué aux problèmes d'optimisation [14], dès qu'un séparateur S_e est affecté, le problème correspondant P_e/A_e est résolu comme un sous-problème indépendant du reste du problème. Cela signifie que son majorant initial est égal à k.

Ainsi, des algorithmes existants tels que BTD ou AND-OR Search n'imposent pas un majorant initial lorsqu'ils résolvent un sous-problème. Bien que cela garantisse effectivement que l'optimum de P_e/A_e sera correctement calculé, cela ne permet pas de disposer d'une coupe initiale efficace. De plus, ce coût optimal, une fois combiné avec le coût des clusters déjà totalement affectés et avec des minorants existants des clusters restants non explorés peut très bien dépasser le coût de la meilleure solution trouvé jusqu'alors, résultant en un effort d'optimisation inutile.

Dans ce cas, il peut être plus efficace de commencer la résolution de P_e/A_e avec un majorant initial non trivial obtenu en prenant la différence entre le majorant associé au cluster père $P_{\textit{Father}(C_e)}$ et un minorant de $P_{\textit{Father}(C_e)}$ sauf P_e. Cependant, cela peut avoir un effet de bord néfaste : après la résolution d'un sous-problème, si une solution avec un coût strictement inférieur au majorant initial est trouvé, alors cette solution est optimale et peut être mémorisée en tant que telle. Mais si aucune solution n'est trouvée, nous avons alors seulement produit un minorant de l'optimum de P_e/A_e. Par la suite, nous mémorisons le minorant et le fait qu'il soit optimal dans LB_{P_e/A_e} et Opt_{P_e/A_e} respectivement, initialement mis à 0 et *faux*.

Si la recherche revient plus tard sur le sous-problème P_e/A_e, et même si un minorant a déjà été calculé, il peut être nécessaire de résoudre à nouveau le sous-problème. Cependant, puisque le minorant mémorisé sera nécessairement amélioré à chaque exploration de P_e/A_e, le nombre de résolutions successives est borné par k (et plus précisément par l'optimum de P_e/A_e).

L'algorithme résultant a la même complexité spatiale que BTD et le nombre de noeuds visités est borné par $O(k.d^{w+1})$ où w est la largeur d'arbre de la décomposition employée. Remarquons que parce que l'algorithme exploite toujours une décomposition arborescente, le nombre de noeuds est aussi dominé par $O(d^h)$ où h est la hauteur d'arbre de la décomposition (ainsi cela reste meilleur que la recherche par séparation et évaluation standard en $O(d^n)$).

4 Cohérence locale et décomposition arborescente

Indépendamment des approches exploitant une décomposition arborescente, l'efficacité de la résolution des WCSP a été largement améliorée ces dernières années par l'exploitation d'extensions des cohérences locales aux fonctions de coûts [24]. Plusieurs cohérences locales de plus en plus fortes ont été définies depuis lors (cohérence souple de noeud et d'arc [24, 16, 20], cohérence d'arc directionnelle complète (DAC/FDAC) [5, 19, 4] et cohérence d'arc existentielle (EDAC) [11]), amenant à des algorithmes de plus en plus efficaces.

Comme dans le cas classique, appliquer un renforcement de cohérence locale sur un problème initial produit en temps polynomial un problème *équivalent*, ç-à-d. définissant la même distribution de coûts pour toutes les affectations complètes, avec des domaines potentiellement plus petits. Cela peut aussi augmenter la valeur du minorant $W_\varnothing$ et des coûts unaires $W_i(a)$ associés aux valeurs des domaines. $W_\varnothing$ définit un minorant fort qui peut être exploité par les algorithmes de séparation et évaluation, tandis que la mise à jour des $W_i(a)$ peut servir à informer des heuristiques de choix de variables et de valeurs.

Dans notre cas, il est important de noter que le renforcement de ces cohérences locales est fait par l'application répétée d'opérations atomiques appelées *arc equivalence preserving transformations* [4]. Si nous considérons deux fonctions de coûts W_{ij} et W_i, une valeur $a \in D_i$ et un coût α, nous pouvons ajouter α à $W_i(a)$ et soustraire α de chaque coût $W_{ij}(a,b)$ pour tout $b \in D_j$. Des calculs arithmétiques simples montrent que la distribution des coûts reste inchangée tandis que les coûts peuvent avoir bougés du niveau binaire au niveau unaire si $\alpha > 0$ (ceci est appelé une Projection) ou du niveau unaire au niveau binaire si $\alpha < 0$ (ceci est appelé une Extension). Lors de ces opérations atomiques, n'importe quel coût au dessus de k, le coût maximum autorisé, peut être considéré comme infini et ainsi ne pas être affecté par la soustraction. Si aucun coût négatif n'apparaît, et si tous les coûts au dessus de k sont mis à k, alors le problème résultant est toujours un problème WCSP valide et équivalent au problème initial.

Le même mécanisme peut être appliqué au niveau unaire pour déplacer des coûts de W_i vers $W_\varnothing$. Enfin, n'importe quelle valeur a telle que $W_i(a) + W_\varnothing$ est égal à k peut être éliminée.

À partir de cette description, on peut déjà voir que même si les opérations atomiques réaliséees lors du renforcement d'une propriété de cohérence locale préservent le coût des affectations complètes, elles peuvent déplacer des coûts entre clusters, invalidant alors les informations précédemment mémorisées.

Projections et extensions binaires Déplacer des coûts entre fonctions de coûts W_{ij} et $W_i(a)$ peut modifier la distribution des coûts (et par là les optima mémorisés) des sous-problèmes. Cela n'intervient que lorsque W_i et W_{ij} sont associés à des sous-problèmes différents, tels que par exemple $W_{72} \in C_1$ et $W_{72,69} \in C_5$ dans la Figure 1, et en particulier lorsque le premier cluster le plus proche de la racine qui contient i, $C_{[i]}$, est un ascendant de $C_{[j]}$. Dans ce cas, la distribution des coûts de $P_{[j]}$ et de tous les sous-problèmes ascendants jusqu'à $P_{[i]}$ (mais ne l'incluant pas) est diminuée ou augmentée (en fonction du signe de α).

Nous mémorisons ces modifications de coûts dans une structure de données spécifique *backtrackable* $\Delta W_i^e(a)$. Il y a un compteur pour chaque valeur de chaque variable dans chaque séparateur (la même variable pouvant apparaître dans plusieurs séparateurs). La complexité spatiale de cette structure est en $O(mnd)$.

Initialement, ΔW est mis à zéro. Lorsqu'une opération atomique est effectuée telle que décrite précédemment, tous les compteurs $\Delta W_i^e(a)$ pour tous les séparateurs S_e, de $S_{[j]}$ jusqu'à $S_{[i]}$ (exclus) sont mis à jour : $\Delta W_i^e(a) := \Delta W_i^e(a) + \alpha$.

Durant la recherche, lorsqu'un séparateur S_e est affecté par une affectation A_e, nous pouvons obtenir le coût total qui a été déplacé hors du sous-problème P_e/A_e en sommant tous les $\Delta W_i^e(a)$ pour toutes les valeurs (i,a) de l'affectation du séparateur A_e. Cette somme sera utilisée pour corriger les minorants mémorisés. Elle est appelée $\overline{\Delta W}_{P_e/A_e}$ par la suite.

Par exemple, dans la Figure 1, nous avons $\overline{\Delta W}_{P_5/A_5} = \Delta W_{57}^5(a) + \Delta W_{72}^5(b)$ avec $A_5 = \{(57,a),(72,b)\}$.

Projection unaire Le renforcement de cohérence locale utilise un mécanisme similaire pour déplacer les coûts des fonctions de coûts unaires W_i vers le minorant du problème $W_\varnothing$. Dans le but d'éviter des problèmes similaires concernant la distribution des coûts, nous répartissons $W_\varnothing$ dans des fonctions de coûts locales $W_\varnothing^e$ associées aux clusters. Lorsqu'un coût unaire W_i est déplacé vers le niveau d'arité zéro, il est déplacer dans $W_\varnothing^{[i]}$. Pour un sous-problème donné P_e, son minorant associé est obtenu en additionnant toutes les informations locales $W_\varnothing^f$ pour tous les clusters C_f dans P_e. Cela définit un minorant local noté

$\overline{W}^e_\varnothing$. Pour le problème complet, ce minorant local est égal au minorant global $W_\varnothing$.

Pour tout sous-problème strict $P_e, e \in \{2, \ldots, m\}$ et toute affectation A_e, nous avons à disposition deux minorants, l'un produit par le mécanisme de mémorisation des minorants ($LB_{P_e/A_e} - \overline{\Delta W}_{P_e/A_e}$) et l'autre produit par l'application de la cohérence locale ($\overline{W}^e_\varnothing$). Dans la suite, nous utilisons le maximum de ces deux bornes, noté $lb(P_e/A_e)$. Nous généralisons la définition précédente à n'importe quel sous-problème $P_e, e \in \{1, \ldots, m\}$ et n'importe quelle affectation partielle courante A : $lb(P_e/A) = W^e_\varnothing + \sum_{C_f \in \mathit{Sons}(C_e)} lb(P_f/A_f)$, qui prend en compte la cohérence locale et, dès que les séparateurs sont complètement affectés, les minorants mémorisés correspondants.

Par exemple, dans la Figure 1, soit $A = \{(57, a), (72, b), (69, c)\}$, alors $lb(P_5/A) = W^5_\varnothing + lb(P_6/A_6) = W^5_\varnothing + \max\{\overline{W}^6_\varnothing, LB_{P_6/A_6} - \overline{\Delta W}_{P_6/A_6}\} = W^5_\varnothing + \max\{W^6_\varnothing + W^7_\varnothing, LB_{P_6/A_6} - \Delta W^6_{69}(c)\}$ avec $A_6 = \{(69, c)\}$.

Filtrage à l'aide de coupes locales Le dernier mécanisme utilisé dans l'application d'une cohérence locale est l'élimination de valeurs ou filtrage. Ici, toute valeur $a \in D_i$ telle que $W_\varnothing + W_i(a)$ est supérieur au majorant global gub (le coût de la meilleure solution connue) peut être éliminée. Pour un sous-problème P_e, nous remplaçons cette condition globale par une condition locale $\overline{W}^e_\varnothing + W_i(a) \geq cub$ où $\overline{W}^e_\varnothing$ et cub sont les minorants et majorants du sous-problème P_e en cours de résolution. Une meilleure règle de filtrage peut être obtenue en prenant en compte les informations provenant de la cohérence locale et des minorants mémorisés d'une façon similaire au calcul de $lb(P_e/A)$ mais en faisant attention à ne pas utiliser LB_{P_f/A_f} si P_f inclut la valeur a à éliminer (pour ne pas compter deux fois le même coût $W_i(a)$ dans $\overline{W}^f_\varnothing$ et dans LB_{P_f/A_f}). Nous appliquons cette règle uniquement sur les variables du sous-problème courant P_e.

En faisant cela, nous assurons que toute valeur éliminée dans un sous-problème P_e est aussi interdite dans tous les sous-problèmes P_f inclus dans P_e. Ainsi l'ensemble des valeurs éliminées pour un sous-problème P_e est encore valide dans tous les sous-problèmes P_f tels que $C_f \in \mathit{Sons}(C_e)$, et toutes les propagations de coûts effectuées durant l'exploration de C_e sont encore valides lors de l'exploration des $C_f \in \mathit{Sons}(C_e)$. La preuve vient du fait que la différence entre les majorants et minorants courants décroît de façon monotone le long d'un chemin allant de la racine à une feuille de l'arbre de décomposition.

Par exemple, dans la Figure 1, soit P_5 le sous-problème courant et $A = \{(57, a), (72, b), (69, c)\}$ l'affectation partielle courante. Toute valeur $u \in D_i$ d'une variable $i \in V_5$ est éliminée si $lb(P_5/A) + W_i(u) \geq cub$. Toute valeur $v \in D_j$ d'une variable $j \in V_6 \cup V_7$ est également éliminée si $\overline{W}^5_\varnothing + W_j(v) \geq cub$.

Remarque sur l'implémentation du filtrage : durant l'exploration du cluster C_e, dès qu'un séparateur S_f de $C_f \in \mathit{Sons}(C_e)$ est affecté, le minorant du sous-problème P_f/A_f obtenu par propagation ($\overline{W}^f_\varnothing$) est comparé à celui obtenu par mémorisation ($LB_{P_f/A_f} - \overline{\Delta W}_{P_f/A_f}$). Si ce second minorant est strictement supérieur au premier, ou s'il est optimal (Opt_{P_f/A_f} vaut vrai), alors le sous-problème P_f/A_f est temporairement déconnecté de la propagation (pas de filtrage des valeurs à l'intérieur) et son minorant est ajouté au minorant courant $\overline{W}^e_\varnothing$ dans le but d'éliminer des valeurs de C_e ou des autres clusters fils de C_e en appliquant simplement la condition locale $\overline{W}^e_\varnothing + W_i(a) \geq cub$.

Filtrage à l'aide de coupes locales et globales À l'intérieur d'un sous-problème, une valeur peut être éliminée pour des raisons locales si $\overline{W}^e_\varnothing + W_i(a) \geq cub$ mais aussi, comme dans la méthode traditionnelle de séparation et évaluation, pour des raisons globales, si $W_\varnothing + W_i(a) \geq gub$. Parce qu'aucune de ces deux conditions n'inclut l'autre, elles peuvent être toutes les deux utiles. Cependant, la condition globale peut couper l'exploration d'un cluster C_e de telle sorte que premièrement nous perdons la garantie que si une solution est trouvée, elle est optimale, mais aussi la garantie qu'un meilleur minorant puisse être produit en cas d'absence de solution. Même s'il est possible d'inférer un minorant valide en collectant les coûts aux feuilles de l'arbre d'exploration du sous-problème courant, le nombre de résolutions successives d'un même sous-problème n'est plus borné par k et la seule borne de complexité sur le nombre total de noeuds visités reste en $O(d^h)$. Lorsque la mémorisation des minorants est employée, cette approche est alors dominée en théorie (en temps et en espace) par les approches précédentes. Puisque le résultat des expérimentations (non reporté ici, voir [9]) n'a pas montré de performances intéressantes, cette approche utilisant la *double coupe* (condition locale et globale) n'est considérée que dans le cas où il n'y a pas de mémorisation de minorants (Pseudo-Tree Search).

Ci-contre, nous présentons le pseudo-code de l'algorithme Lc-BTD$^+$ qui exploite une décomposition arborescente et maintient une propriété de cohérence locale donnée par le paramètre *Lc*. Cet algorithme utilise notre amélioration des majorants initiaux (ligne 1), le filtrage des valeurs fondé sur les coupes locales et de façon optionnelle une mémorisation des minorants (lignes 2 et 3). L'appel initial est Lc-BTD$^+$($\emptyset$,C_1,V_1,k). Cela suppose un problème déjà rendu Lc-cohérent. L'algorithme retourne l'optimum du problème. La fonction pop(S) retourne un élément de S et retire cet élément de S.

Lc-BTD$^+$ a la complexité spatiale au pire cas habituelle

```
Function Lc-BTD+(A,C_e,V,cub) : integer
  if (V = ∅) then
    S := Sons(C_e) ;
    /* Résouds tous les clusters fils dont l'optimum est inconnu */ ;
    while (S ≠ ∅ and lb(P_e/A) < cub) do
      C_f :=pop(S) /* Choix d'un cluster fils */ ;
      if (not(Opt_{P_f/A_f})) then
1       cub' := cub − lb(P_e/A) + lb(P_f/A_f) ;
        clb' :=Lc-BTD+(A, C_f, V_f, cub') ;
2       LB_{P_f/A_f} := clb' + ΔW_{P_f/A_f} ;
3       Opt_{P_f/A_f} := (clb' < cub') ;
    return lb(P_e/A) ;
  else
    i :=pop(V) /* Choix d'une variable non affectée */ ;
    d := D_i ;
    /* Énumération des valeurs du domaine de i */ ;
    while (d ≠ ∅ and lb(P_e/A) < cub) do
      a :=pop(d) /*Choix non-déterministe de valeur*/ ;
      Affecte a à i et applique Lc sur P_e/A_e;
      if (lb(P_e/A ∪ {(i, a)}) < cub) then
        cub := min(cub,Lc-BTD+(A ∪ {(i, a)}, C_e, V, cub));
    return cub ;
```

des méthodes exploitant une décomposition arborescente, c'est à dire exponentielle dans la taille du plus grand séparateur. Parce qu'il utilise un majorant initial amélioré présenté précédemment, le nombre de noeuds explorés est borné par $O(k.d^{w+1})$. Grâce à la coupe du principe de séparation et évaluation, il est possible que seule une petite fraction de l'espace théorique soit utilisé. Notre implémentation utilise donc une structure de donnée creuse (table de hachage) pour mémoriser les informations des minorants.

5 Résultats expérimentaux

Dans cette section, nous effectuons une comparaison empirique de l'algorithme Lc-BTD$^+$ avec l'algorithme BTD standard maintenant une propriété de cohérence locale de type Forward Checking (FC-BTD) [14], l'algorithme de recherche arborescente par séparation et évaluation en profondeur d'abord (MFDAC) [19], et enfin Variable Elimination (VE) [12], un algorithme de programmation dynamique ayant une complexité spatiale et temporelle en $O(d^{w+1})$, sur des instances binaires (fonctions de coûts d'arité au plus deux) de problèmes aléatoires et de problèmes issus du monde réel.

Plusieurs version de notre algorithme peuvent être considérées : sans mémorisation (lignes 2 et 3 supprimées), nous obtenons une variante de Pseudo-Tree Search, noté FDAC-PTS, amélioré par un maintien de cohérence locale. Cet algorithme est similaire à l'algorithme AOMF-DAC [22]. Dans ce cas, les coupes locales et globales sont utilisées pour éliminer des valeurs. Pour la cohérence locale Lc, nous avons testé la cohérence de noeuds (NC-BTD+) et la cohérence d'arc directionnelle complète (FDAC-BTD+).

Tous les algorithmes sont implémentés dans le langage C et sont disponibles dans un logiciel d'optimisation en open source appelé TOOLBAR (`http://-carlit.toulouse.inra.fr/cgi-bin/awki.cgi/SoftCSP`) qui accepte des fonctions de coûts binaires ou d'arité supérieure définies en extension par un ensemble de n-uplets avec leur coût associé. Les expérimentations ont été réalisées sur un Xeon $2,4$ GHz avec 4 GB.

Instances générées aléatoirement Les classiques problèmes aléatoires non structurés ne présentent pas d'intérêt pour des approches exploitant une décomposition arborescente, car pour ces problèmes la largeur d'arbre est souvent proche du nombre total de variables. Nous avons construit un générateur de problèmes Max-CSP (chaque violation d'une contrainte a un coût de 1) aléatoires structurés par un arbre binaire de cliques. Les paramètres du modèle sont (w, s, h', d, t) où $w + 1$ est la taille des cliques, s la taille des séparateurs, h' la hauteur de l'arbre de cliques, d la taille des domaines et t la dureté des contraintes (pourcentage de n-uplets interdits). Le nombre total de variables est égal à $n = s + (w + 1 - s)(2^{h'} - 1)$. Notre méthode produisant une décomposition arborescente identique à l'arbre de cliques, aboutit à une largeur d'arbre égale à w et une hauteur d'arbre égale à $h = h'(w + 1 - s) + s$. Nous avons générés et résolus des arbres de cliques avec $(w = 9, h' = 3, d = 5)$ pour différentes tailles de séparateurs $s \in \{2, 5, 7\}$ et différentes duretés des contraintes $t \in [30, 90]\%$ (uniquement des instances sur-contraintes). Les résultats reportés sont des moyennes sur 50 instances. Tous les algorithmes (excepté VE) utilisent l'heuristique dynamique de choix de variables *domaine/degré* (localement à l'intérieur des clusters pour les méthodes exploitant une décomposition arborescente). L'heuristique de choix de valeurs consiste à sélectionner la valeur ayant le coût unaire W_i le plus grand en premier. L'ordre des variables pour DAC est l'ordre lexicographique. L'ordre d'élimination de variables utilisé par VE et pour construire les décompositions arborescentes est produit par l'heuristique Maximum Cardinality Search (MCS) [27] qui a une complexité linéaire en $O(n + |E|)$, où E est l'ensemble des arêtes du graphe de contraintes binaires. La racine de la décomposition est choisie de façon à minimiser la hauteur d'arbre. Les temps CPU relevés ne prennent pas en compte le temps pour calculer une décomposition arborescente (temps toujours inférieur à 10 ms sur toutes nos instances). Le temps est limité à 5 minutes par instance (les instances non résolues dans le temps imparti comptent pour un temps de

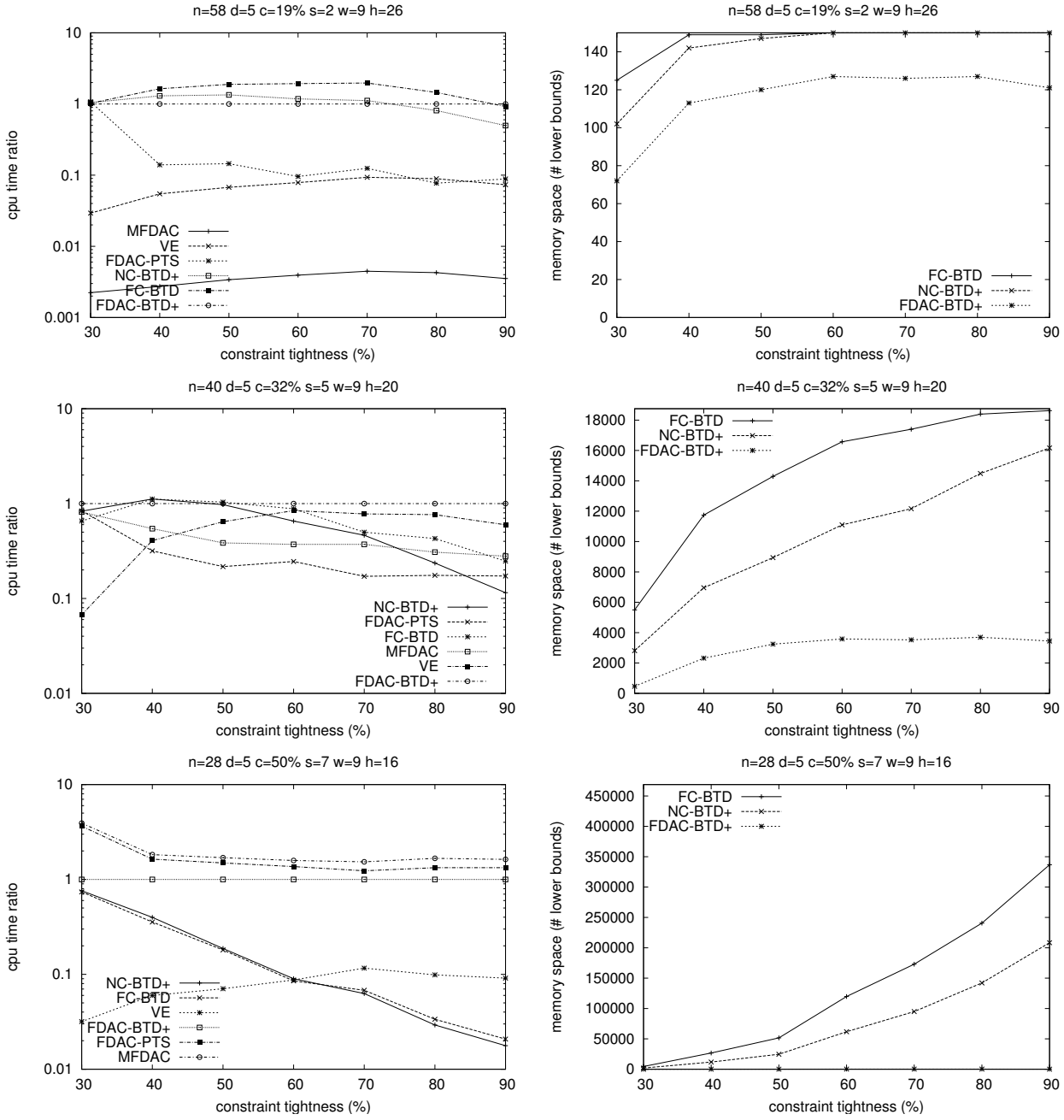

Figure 2: Ratio de temps CPU entre FDAC-BTD+ et les autres méthodes (échelle logarithmique pour l'axe des y) et nombre de minorants mémorisés (échelle linéaire pour l'axe des y dont la valeur maximum correspond à la borne théorique maximum) pour trouver et prouver l'optimalité de problèmes aléatoires sous forme d'arbres binaires de cliques. Taille des séparateurs s égale à 2, 5 et 7 (figures de haut en bas). Les algorithmes sont triés par efficacité croissante du pire (en haut) au meilleur (en bas) à la dureté des contraintes $t = 90\%$. n, d, c, w et h sont le nombre de variables, la taille des domaines, la connectivité du graphe des contraintes, la largeur d'arbre et la hauteur d'arbre respectivement.

RLFAP	SUB_1		SUB_4		SCEN-06	
optimum	2669		3230		3389	
n, d, w, h	$14, 44, 13, 14$		$22, 44, 19, 21$		$100, 44, 19, 67$	
Méthode	temps	#LB	temps	#LB	temps	#LB
FC-BTD	1197	0	-	0	-	-
NC-BTD+	490	0	-	0	-	-
FDAC-BTD+	14	0	929	0	10.309	326
FDAC-PTS	14	n/a	851	n/a	-	n/a
MFDAC	14	n/a	984	n/a	-	n/a

Table 1: Temps CPU en secondes (le symbole"-" signifie que la limite des 10 heures de calcul a été atteinte) et mémoire en nombre de minorants mémorisés (#LB) pour prouver l'optimalité d'instances RLFAP.

résolution de 5 minutes). La figure 2 donne le ratio de temps CPU entre FDAC-BTD+ et les autres méthodes et l'occupation mémoire en nombre de minorants mémorisés pour les méthodes avec mémorisation (sans compter les répétitions pour un même sous-problème).

Pour des cohérences locales faibles telles que FC et NC (qui produisent le même minorant), notre majorant initial amélioré permet de réduire l'occupation mémoire au prix de temps de calcul plus importants. Avec des cohérences locales plus fortes telles que FDAC, FDAC-BTD+ est toujours parmi les algorithmes les plus performants et peut accélérer la recherche de deux ordres de magnitude par rapport à FC-BTD dans le cas d'une large taille de séparateurs ($\frac{1}{3}$ des variables sont dans le cluster racine pour $s = 7$) et d'une forte dureté des contraintes (voir [17] pour une comparaison empirique entre Forward Checking et des cohérences locales exploitant les contraintes futures). Pour des tailles petite ou moyenne de séparateurs, la mémorisation des minorants semble être un élément crucial pour l'efficacité des méthodes, FDAC-PTS étant même dominé par FC-BTD et NC-BTD+ dans ce cas. MFDAC est incapable de résoudre des instances avec un grand nombre de variables dans le cas de séparateurs de petite taille. Enfin, VE ayant une complexité indépendante de la dureté des contraintes, il nécessite un temps et une taille de mémoire constant (bornés par $d^{w+1} = 5^{10} = 9,8\ 10^6$). Parmi toutes les instances résolues, le nombre maximum de résolutions successives d'un même sous-problème P_e/A_e n'a jamais dépassé 11 pour FDAC-BTD+ et 17 pour NC-BTD+, bien loin de la borne théorique définie par le coût optimal le plus grand, ici égal à 202.

Instances issues d'un problème du monde réel Le problème Radio Link Frequency Assignment Problem (RLFAP) [3] consiste à affecter des fréquences à un ensemble de liens radio de telle façon que les liens puissent fonctionner ensemble sans interférence notable. Certaines instances RLFAP peuvent être naturellement traduites en CSP binaires pondérés. Nous étudions plus particulièrement l'instance SCEN-06 et ses sous-instances SUB_1 et SUB_4 ($SUB_1 \subset SUB_4 \subset$ SCEN-06). Par rapport aux réglages précédents, nous utilisons le coût de la meilleure solution connue (optimal) comme majorant global initial. Un prétraitement de l'arbre de décomposition produit par MCS (dont le temps de calcul est aussi inférieur à 10 ms pour SCEN-06) est effectué pour fusionner les paires de clusters adjacents ayant une taille de séparateur supérieure strictement à 3. L'heuristique dynamique de choix de variables est *2-sided Jeroslow-like* décrite dans [7]. Le cluster racine est celui qui maximise le produit des tailles de domaines de ses variables. De cette manière, les clusters racines de SCEN-06 et SUB_4 sont identiques (20 variables) et contiennent la partie difficile du problème. La limite sur le temps de calcul est de 10 heures. Les mêmes statistiques sur le temps en secondes et la mémoire en nombre de minorants mémorisés (#LB) sont indiquées dans le Tableau 1.

À nouveau, FDAC-BTD+ s'avère être la méthode la plus robuste. Légèrement dominée par FDAC-PTS sur l'instance SUB_4, il est le seul algorithme capable de prouver l'optimalité de l'instance complète SCEN-06 en moins de 10 heures (au total 3 heures de calcul). La mémorisation, même si elle est assez réduite (326 minorants mémorisés pour un borne théorique d'environ un million(M), ces minorants sont utilisés 5.5M de fois durant la recherche qui visite 16M de noeuds), c'est la seule différence (excepté aussi la politique de filtrage) par rapport à l'algorithme FDAC-PTS qui n'a pas résolu le problème dans la limite des 10 heures de calcul (symbole "-") mais a terminé en environ deux jours de calcul. Variable Elimination n'a pu résoudre aucune instance à cause de la limite sur l'espace mémoire de 4 GB.

Comparé aux travaux existants, dans [14], FC-BTD sans majorant global initial a résolu l'instance SUB_4 en 122.933 secondes (34 heures). AOMFDAC utilisant un ordre statique de choix de variables a résolu la même instance en 47.115 secondes (13 heures)[22]. Ces deux résultats expérimentaux ont été obtenus sur un Pentium IV $2,4$ GHz mais ont utilisé un arbre de décomposition différent du notre. Premièrement résolu avec une approche fondée sur un partitionnement du problème (SUB_4 étant la plus grande sous-composante) [10], l'instance complète SCEN-06 a été aussi résolue par un algorithme dédié de programmation dynamique en 27.102 secondes (7,5 heures) sur une station de travail DEC 2100 A500MP [15] (fréquence de processeur supérieure à 200 Mhz, d'où un temps converti d'environ $\frac{27.102*200}{2400} = 2.258$ secondes). Cependant notre approche est davantage générique, nous n'utilisons pas de prétraitement spécifique aux problème d'allocations de fréquence et la mémoire consommée est plus réduite (326 comparé à environ un million [15]).

6 Conclusion

Dans ce papier, nous avons proposé une recherche arborescente de type séparation et évaluation en profondeur d'abord qui exploite une décomposition arborescente et une propriété de cohérence locale. Parmi les approches potentielles exploitant une décomposition arborescente, nous avons trouvé que l'approche la plus prometteuse et aussi la plus robuste consiste à combiner des majorants initiaux améliorés, une cohérence locale forte, des minorants mémorisés *corrigés* et un filtrage des valeurs utilisant une coupe locale. L'algorithme résultant sacrifie un peu les bornes de complexité théorique pour une meilleure efficacité en pratique (en temps et en espace), ce qui a été montré sur des instances aléatoires et des instances issues du monde réel.

Nous avons présenté notre algorithme dans le cadre des CSP pondérés binaires. Notre approche est aussi applicable à des contraintes souples non binaires, ouvrant la porte à d'autres domaines tels que Max-SAT et MPE dans les Réseaux Bayésiens.

Remerciements Nous remercions les développeurs de TOOLBAR et en particulier Sylvain Bouveret pour son travail sur les méthodes de décomposition arborescente.

References

[1] F. Bacchus, S. Dalmao, and T. Pitassi. Value elimination: Bayesian inference via backtracking search. In *Proc. of UAI-03*, pages 20–28, Acapulco, Mexico, 2003.

[2] H. Bodlaender. Discovering treewidth. In *Theory and Practive of Computer Science - SOFSEM'2005*, pages 1–16, 2005.

[3] B. Cabon, S. de Givry, L. Lobjois, T. Schiex, and J.P. Warners. Radio link frequency assignment. *Constraints*, 4:79–89, 1999.

[4] M. Cooper and T. Schiex. Arc consistency for soft constraints. *Artificial Intelligence*, 154:199–227, 2004.

[5] M.C. Cooper. Reduction operations in fuzzy or valued constraint satisfaction. *Fuz. Sets and Sys.*, 134(3):311 – 342, 2003.

[6] A. Darwiche. Recursive Conditioning. *Artificial Intelligence*, 126(1-2):5–41, 2001.

[7] S. de Givry, J. Larrosa, P. Meseguer, and T. Schiex. Solving max-sat as weighted csp. In *Proc. of CP-03*, pages 363–376, Kinsale, Country Cork, Ireland, 2003.

[8] S. de Givry, I. Palhiere, Z. Vitezica, and T. Schiex. Mendelian error detection in complex pedigree using weighted constraint satisfaction techniques. In *ICLP-05 workshop on Constraint Based Methods for Bioinformatics*, Sitges, Spain, 2005.

[9] S. de Givry, T. Schiex, and G. Verfaillie. Combining tree decomposition and local consistency in max-csps. In *CP-2005 workshop on Preferences and Soft Constraints*, Sitges,Spain, 2005.

[10] S. de Givry, G. Verfaillie, and T. Schiex. Bounding the Optimum of Constraint Optimization Problems. In *Proc. of CP-97*, pages 405–419, Schloss Hagenberg, Austria, October 29 - November 1 1997.

[11] S. de Givry, M. Zytnicki, F. Heras, and J. Larrosa. Existential arc consistency: Getting closer to full arc consistency in weighted csps. In *Proc. of IJCAI-05*, pages 84–89, Edinburgh, Scotland, 2005.

[12] R. Dechter. Bucket elimination: A unifying framework for reasoning. *Artificial Intelligence*, 113(1–2):41–85, 1999.

[13] E. C. Freuder and M. J. Quinn. Taking advantage of stable sets of variables in constraint satisfaction problems. In *Proc. of IJCAI-85*, pages 1076–1078, Los Angeles, CA, 1985.

[14] P. Jégou and C. Terrioux. Decomposition and good recording. In *Proc. of ECAI-04*, pages 196–200, Valencia, Spain, 2004.

[15] A. Koster. *Frequency assignment: Models and Algorithms*. PhD thesis, Maastricht, The Netherlands, 1999.

[16] J. Larrosa. On arc and node consistency in weighted CSP. In *Proc. AAAI-02*, pages 48–53, Edmondton, (CA), 2002.

[17] J. Larrosa and P. Meseguer. Phase transition in max-csp. In *Proc. of ECAI-96*, pages 190–194, Budapest, Hungary, 1996.

[18] J. Larrosa, P. Meseguer, and M. Sanchez. Pseudo-tree search with soft constraints. In *Proc. of ECAI-02*, pages 131–135, Lyon, France, 2002.

[19] J. Larrosa and T. Schiex. In the quest of the best form of local consistency for weighted CSP. In *Proc. of IJCAI-03*, pages 239–244, Acapulco, Mexico, 2003.

[20] J. Larrosa and T. Schiex. Solving Weighted CSP by Maintaining Arc-consistency. *Artificial Intelligence*, 159(1-2):1–26, 2004.

[21] R. Marinescu and R. Dechter. Advances in and/or branch-and-bound search for constraint optimization. In *CP-2005 workshop on Preferences and Soft Constraints*, Sitges, Spain, 2005.

[22] R. Marinescu and R. Dechter. And/or branch-and-bound for graphical models. In *Proc. of IJCAI-05*, pages 224–229, Edinburgh, Scotland, 2005.

[23] T. Sandholm. An algorithm for optimal winner determination in combinatorial auctions. In *Proc. of IJCAI-99*, pages 542–547, Stockholm, Sweden, 1999.

[24] T. Schiex. Arc consistency for soft constraints. In *Proc. of CP-00*, volume 1894 of *LNCS*, pages 411–424, Singapore, 2000.

[25] T. Schiex, H. Fargier, and G. Verfaillie. Valued constraint satisfaction problems: hard and easy problems. In *Proc. of IJCAI-95*, pages 631–637, Montréal, Canada, August 1995.

[26] D. Strickland, E. Barnes, and J. Sokol. Optimal protein structure alignment using maximum cliques. *Operations research*, 53(3):389–402, 2005.

[27] R. Tarjan and M. Yannakakis. Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs and selectively reduce acyclic hypergraphs. *SIAM J. Comput.*, 13(3):566–579, 1984.

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