Algorithme des Poupées Russes exploitant une décomposition arborescente

M. Sanchez, David Allouche, Simon de Givry, Thomas Schiex

Poupées russes et décomposition arborescente

M. Sanchez, D. Allouche, S. de Givry, T. Schiex
UBIA, UR 875, INRA, F-31320 Castanet Tolosan, France
{msanchez,allouche,degivry,tschiex}@toulouse.inra.fr

Abstract

Les problèmes d'optimisation dans les modèles graphiques ont été étudiés dans différents formalismes de l'intelligence artificielle tels que les CSP pondérés (weighted CSP ou WCSP), le maximum de satisfiabilité (MaxSAT) ou dans les réseaux probabilistes (réseaux Bayésiens et champs markoviens). En identifiant des sous-problèmes conditionnellement indépendants,qui sont résolus de façon indépendante et dont l'optimum est mémorisé, il est possible de rendre les algorithmes de type Séparation-Évaluation (Branch and Bound) asymptotiquement plus efficaces. Mais la localité des bornes induites par la décomposition affaiblit les effets réels de ce résultat asymptotique car de nombreux sous-problèmes, ne participant pas à la construction d'une solution optimale, sont résolus à l'optimum inutilement.

En s'inspirant de l'algorithme des poupées russes (RDS pour Russian Doll Search), une approche possible pour surmonter cette faiblesse consiste à (récursivement) résoudre une relaxation de chacun des sous-problèmes pour obtenir des bornes renforcées. L'algorithme ainsi défini généralise à la fois l'algorithme RDS mais aussi les algorithmes de types Branch and Bound exploitant une décomposition arborescente tels que BTD ou AND-OR Branch and Bound. Nous étudions son efficacité sur différents problèmes et fermons une instance très dure de problème d'affectation de fréquences ouverte depuis plus de 10 ans.

1 Introduction

Le traitement de modèles graphiques est un problème central de l'intelligence artificielle. L'optimisation d'une combinaison de fonctions de coût local est en particulier étudiée dans les réseaux de contraintes valués [21] . Elle permets de capturer des problèmes tels que le problème MaxSAT pondéré, CSP pondérés, recherche d'une explication de probabilité maximum (MPE pour Maximum Probability Explanation) dans les réseaux Bayésiens et les champs markoviens. Ses applications directes sont nombreuses notamment en *affectation de ressources* [23, 2] et en bioinformatique [19].

Ces dernières années, afin de résoudre des problèmes de satisfaction, d'optimisation ou de comptage, différents algorithmes de recherche arborescente exploitant le couplage d'une décomposition arborescente du problème avec une propagation des informations dures du réseau ont été proposés. Cette classe d'algorithme couvre l'algorithme de Recursive Conditioning [7], l'algorithme Backtrack bounded by Tree Decomposition (BTD) [22] et l'algorithme AND-OR graph search [17], tous héritiers de l'algorithme de Pseudo-Tree Search [11]. En comparaison avec des algorithmes de recherche arborescente traditionnels, ces algorithmes offrent des complexités asymptotiques améliorées, seulement exponentielles dans la largeur d'arbre du graphe du problème. Ce gain s'obtient toutefois au prix d'une restriction dans l'ordre d'affectation des variables (voir [13]).

Dans le contexte de l'optimisation (sans perte de généralité, nous considérons des problèmes de minimisation), l'exploitation d'une décomposition du problème a pour effet secondaire la perte de bornes globales. Dans les algorithmes Branch and Bound en profondeur d'abord traditionnels (DFBB pour Depth First Branch and , Bound), l'élagage se produit dès que le coût de la meilleure solution connue (le majorant) rencontre un minorant spécifique, habituellement calculé à partir d'une relaxation du problème global. Lorsqu'une décomposition arborescente est exploitée et qu'une affectation des variables connectant un sous-problème au reste du problème rend ce sous-problème indépendant, une solution optimale du sous-problème (conditionné par cette affectation) est recherchée. Ce calcul de solution optimale est coûteux, et souvent inutile car la solution optimale du sous-problème obtenue ne fait pas nécessairement partie de la solution optimale globale (du fait du reste du problème). Pour limiter ce comportement de "trashing" désagréable, il est possible

d'informer l'algorithme en charge de la recherche d'une solution optimale qu'il doit absolument produire une solution suffisamment bonne pour pouvoir s'intégrer dans une solution globale ou sinon s'arrêter dès qu'il prouve que c'est infaisable. Ce *majorant initial* est obtenu en soustrayant du majorant global une combinaison des minorants disponibles sur le reste du problème.

Dans cet article, nous avons tenté d'améliorer l'algorithme décrit dans [9] qui introduit des minorants obtenus par un filtrage *cohérence locale pondérée* [6] dans l'algorithme BTD. Dans l'objectif d'une meilleure prise en compte de la contribution du problème global dans la définition des majorants locaux, nous remplaçons le moteur de type "Branch and Bound" utilisé dans BTD par une algorithme de type "Poupées Russes" [23]. L'algorithme RDS est étendu afin de pouvoir exploiter une décomposition arborescente et un niveau de cohérence local plus élevé. Dans l'algorithme RDS-BTD ainsi défini, des relaxations des sous-problèmes conditionnellement indépendants identifiés par la décomposition arborescente sont résolues de façon récursive. Les coûts des solutions optimales de ces relaxations définissent de puissants minorants et fournissent donc des majorants locaux améliorés, permettant ainsi de réduire de façon drastique le comportement de "trashing" observé.

Dans la suite de l'article, nous introduisons d'abord le cadre des CSP pondérés, l'algorithme BTD et l'algorithme RDS. Nous présentons ensuite notre algorithme générique qui peut se spécialiser en BTD ou en RDS. Enfin, l'algorithme RDS-BTD est évalué sur différentes instances de problèmes réels issues du domaine de l'affectation de fréquences et de la sélection de "Tag SNP" (en bioinformatique).

2 Weighted Constraint Satisfaction Problem

Un CSP pondéré (Weighted CSP, WCSP) est un quadruplet (X, D, W, m). X et D sont des ensembles de n variables et domaines, comme dans les CSP classiques. Le domaine de la variable i est noté D_i, avec une taille de domaine maximum notée d. Étant donné un sous-ensemble de variables $S \subseteq X$, on note $\ell(S)$ l'ensemble des n-uplets (tuples) définis sur S. W est un ensemble de fonctions de coût. Chaque fonction de coût (ou contrainte molle) w_S de W est définie sur un ensemble de variables S appelé sa portée. Cette dernière est supposée différent pour chaque fonction de coût. Une fonction de coût w_S assigne un coût à chaque affectation des variables de S i.e. : $w_S : \ell(S) \rightarrow [0, m]$. L'ensemble des coûts possibles est $[0, m]$ où $m \in \{1, \ldots, +\infty\}$ représente un coût intolérable. Les coûts sont combinés par l'addition bornée $\oplus$, définie par $a \oplus b = \min\{m, a+b\}$, et comparés entre eux via $\geq$. Notez que le coût intolérable m peut être fini ou infini et qu'un coût b peut être soustrait d'un coût a plus large en utilisant l'opération $\ominus$ où $a \ominus b$ est égal à $(a - b)$ si $a \neq m$. Sinon, il est égal à m.

Pour les fonctions de coût binaires et unaires, nous utilisons des notations simplifiées : une fonction de coût binaire entre les variables i et j est notée w_{ij}. Une fonction de coût unaire sur la variable i est notée w_i. Nous faisons l'hypothèse qu'il existe pour chaque variable, une fonction de coût unaire w_i ainsi qu'une fonction de coût d'arité nulle notée $w_\varnothing$ correspondant à un coût constant payé par toutes affectations.

Le coût d'une affectation complète $t \in \ell(X)$ dans un problème $P = (X, D, W, m)$ est égal à $Val_P(t) = \bigoplus_{w_S \in W} w_S(t[S])$ où $t[S]$ représente la projection habituelle d'un tuple sur l'ensemble de variables S. La minimisation de $Val_P(t)$ défini un problème d'optimisation avec un problème de décision associé qui est NP-complet.

L'établissement d'une cohérence locale donnée sur un problème P consiste à transformer $P = (X, D, W, m)$ en un problème $P' = (X, D, W', m)$ qui est équivalent à P ($Val_P = Val_{P'}$) et qui satisfait la propriété de cohérence locale considérée. Ce processus peut augmenter $w_\varnothing$ et fournir ainsi un minorant amélioré du coût optimum. Il s'appuie sur l'application de transformations préservant l'équivalence (EPT) qui déplace les coûts entre des portées différentes [20, 6, 10, 5].

3 Décompositions arborescentes et algorithme DFBB

Une décomposition arborescente d'un WCSP connexe est définie par un arbre (C, T). Chaque élément C_e de l'ensemble $C = \{C_1, \ldots, C_k\}$ des sommets de l'arbre est appelé un "cluster". Il est formé d'un sous-ensemble de variables ($C_e \subset X$). T est un ensemble d'arêtes définissant un graphe acyclique connexe (un arbre) sur C. L'ensemble des clusters C doit couvrir toutes les variables ($\bigcup_{C_e \in C} C_e = X$) et toutes les fonctions de coût ($\forall w_S \in W, \exists C_e \in C$ *t.q.* $S \subset C_e$). De plus, si une variable i apparaît dans deux clusters C_e et C_g, i doit aussi apparaître dans tout cluster C_f sur l'unique chemin reliant C_e et C_g dans T.

Pour un WCSP donné, nous considérons une décomposition arborescente enracinée (C, T) définie par une racine notée C_1. Dans T, le père (resp. les fils) d'un cluster C_e est noté $Père(C_e)$ (resp. $Fils(C_e)$). Le séparateur de C_e est l'ensemble $S_e = C_e \cap Père(C_e)$. Les variables *propres* d'un cluster C_e est l'ensemble $V_e = C_e \setminus S_e$.

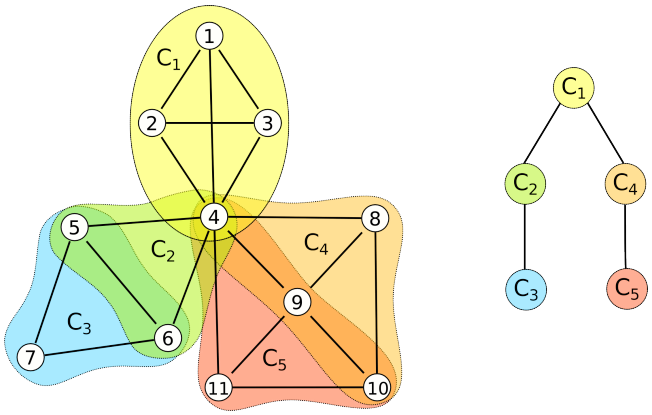

FIG. 1 – Le graphe d'un WCSP et une décomposition arborescente.

La propriété essentielle d'une décomposition arborescente est que l'affectation d'un séparateur S_e coupe le problème initial en (au moins) deux sous-problèmes qui peuvent être résolus indépendemment l'un de l'autre. Le premier sous problème, noté P_e, est défini par les variables de C_e ainsi que celles de tous les clusters descendants dans T. Ses fonctions de coût sont celles qui impliquent *au moins* une variable propre de ces clusters. Les fonctions de coût restantes, avec les variables qu'elles impliquent forment le sous-problème restant.

Exemple 1 *Considérons le MaxCSP de la figure 1. Il a onze variables avec deux valeurs (a, b) par domaine. Les arêtes représentent des contraintes de différence binaires entre les deux variables ($w_{ij}(a, a) = w_{ij}(b, b) = 1, w_{ij}(a, b) = w_{ij}(b, a) = 0$). Dans l'ordre lexicographique, $(a, b, b, a, b, b, a, b, b, a, b)$ représente une solution optimale avec un coût de optimal de 5. Une décomposition arborescente enracinée en C_1 formée des clusters $C_1 = \{1, 2, 3, 4\}, C_2 = \{4, 5, 6\}, C_3 = \{5, 6, 7\}, C_4 = \{4, 8, 9, 10\}$, et $C_5 = \{4, 9, 10, 11\}$ est présentée sur le côté droit de la figure 1. C_1 a pour fils $\{C_2, C_4\}$, le séparateur de C_3 avec son père C_2 est $S_3 = \{5, 6\}$, l'ensemble des variables propres de C_3 est $V_3 = \{7\}$. Notez que la variable 4 est une variable propre de C_1 et appartient aux séparateurs S_2, S_4 et S_5. Le sous-problème P_3 a pour variables $\{5, 6, 7\}$ et comme fonctions de coûts $\{w_{5,7}, w_{6,7}, w_7\}$. P_1 est le problème complet.*

Un algorithme de recherche peut exploiter la propriété précédente pour autant qu'un ordre des variables adapté soit utilisé : les variables d'un cluster C_e quelconque doivent être instanciées avant les variables restantes dans ses clusters fils. Dans ce cas, pour tout cluster $C_f \in \mathit{Fils}(C_e)$, une fois le séparateur S_f affecté, le sous-problème P_f conditionné par l'affectation courante A_f de S_f (notée P_f/A_f) peut être résolue à l'optimalité indépendemment du reste du problème. Le *coût optimal* de P_f/A_f peut alors être enregistré avec pour conséquence qu'il ne sera jamais plus résolu pour cette affectation du séparateur S_f. Ainsi, des algorithmes tels que BTD ou AND-OR graph search sont capables d'avoir une complexité temporelle exponentielle seulement dans la taille du plus grand cluster.

Dans [9], l'algorithme BTD exploite de minorants produits par le maintien d'une propriété de cohérence locale pondérée. Pour tout cluster C_e, une fonction de coût constante (d'arité nulle) spécifique au cluster (notée $w^e_\varnothing$) est utilisée. Elle peut être incrémentée par le mécanisme de filtrage par cohérence locale et ainsi fournir pour chaque cluster un minorant du coût d'une solution optimale. Au delà d'un meilleur élagage, ceci permet également d'éviter de résoudre systématiquement les sous-problèmes à l'optimalité en transmettant une exigence de coût maximum (un majorant) pour chaque sous-problème P_f. Pour P_1, au départ, le coût de la meilleure solution connue définit le majorant initial cub_1 . Il faut ensuite améliorer ce coût. Si l'on considère un cluster arbitraire C_f, fils de C_e avec un majorant associé cub_e, le majorant du sous-problème P_f est obtenu est *en soustrayant* de cub_e les minorants associés à tous les autres clusters fils de C_e. Ainsi, des minorants plus forts sur les autres sous-problèmes induit un majorant plus fort pour le problème courant.

Du fait de ce majorant initial, l'optimum de P_f n'est pas nécessairement calculé mais un minorant est toujours obtenu. Ce minorant et son éventuelle optimalité sont enregistrés dans LB_{P_f/A_f} et Opt_{P_f/A_f} . Ils sont initialement fixés respectivement à 0 et *faux*.

Pour un élagage amélioré et de meilleurs majorants, BTD utilise le maximum du minorant $w^e_\varnothing$ (fourni par la cohérence locale) et du minorant enregistré (LB_{P_f/A_f}). Les minorants enregistrés ne peuvent être utilisés que lorsque le séparateur S_f

est totalement instancié par l'affectation courante A. Ceci permet récursivement de définir le minorant utilisé dans [9] comme $lb(P_e/A) = w^e_\varnothing \oplus \bigoplus_{C_f \in Fils(C_e)} \max(lb(P_f/A), LB_{P_f/A_f})$.

Exemple 2 *Dans l'exemple 1, les variables* $\{1,2,3,4\}$ *de* C_1 *sont affectées en premier, en utilisant par exemple l'heuristique d'ordonnancement dynamique* min domain/max degree. *Soit* $A = \{(4,a),(1,a),(2,b),(3,b)\}$ *l'affectation courante. Le filtrage par la cohérence locale EDAC [10] sur* P_1/A *produit* $w^1_\varnothing = 2, w^2_\varnothing = w^4_\varnothing = 1, w^3_\varnothing = w^5_\varnothing = 0$, *induisant* $lb(P_1/A) = \bigoplus_{C_e \in C} w^e_\varnothing = 4$ *étant donné qu'aucun minorant n'a encore été enregistré.*

Ensuite, le sous-problème $P_2/\{(4,a)\}$ *et* $P_4/\{(4,a)\}$ *sont résolus de façon indépendante et les solutions optimales correspondantes sont mémorisées dans* $LB_{P_2/\{(4,a)\}} = 1$, $LB_{P_4/\{(4,a)\}} = 2$, $Opt_{P_2/\{(4,a)\}} = Opt_{P_4/\{(4,a)\}} =$ *vrai (aucun solution n'étant connue, aucun majorant n'est défini). Une première affectation complète de coût* $w^1_\varnothing \oplus LB_{P_2/\{(4,a)\}} \oplus LB_{P_4/\{(4,a)\}} = 5$ *est obtenue.*

L'algorithme BTD ainsi défini a une complexité temporelle qui, dans le pire des cas, est exponentielle dans la taille du plus grand cluster moins un, un nombre aussi appelé largeur d'arbre de la décomposition arborescente (C,T). Sa complexité spatiale est, dans le pire des cas, exponentielle en s où $s = \max_{C_e \in C} |S_e|$, taille du plus grand séparateur [9].

4 Poupées russes et décomposition arborescentes

L'algorithme original des poupées russes (RDS) [23], a été introduit en 1996, alors qu'aucune cohérence locale pour les CSP pondérés n'avait été définie. Son mécanisme de base a été élaboré pour construire des minorants forts à partir de minorants plus faibles en utilisant une recherche arborescente. L'application de RDS consiste à résoudre n sous-problèmes imbriqués d'un problème initial P avec n variables. Étant donné un ordre des variables statique, RDS commence par résoudre le sous-problème défini par la dernière variable seule. Ensuite, il inclut la variable précédente dans l'ordre et résout le sous-problème avec deux variables. Ce processus est répété jusqu'à la résolution compléte du problème. Chacun des sous-problèmes est résolu avec un algorithme de type DFBB utilisant un ordre statique des variables suivant l'ordre de la décomposition en sous-problèmes imbriqués. Le minorant amélioré utilisé dans l'algorithme DFBB est obtenu en combinant le minorant faible fourni par "Partial Forward Checking" (similaire au filtrage par cohérence de nœud [15]) avec l'optimum du sous-problème résolu à l'étape précédente par RDS.

Nous nous proposons d'exploiter le principe de RDS en s'appuyant sur une décomposition arborescente (RDS-BTD). La principale différence avec RDS réside dans le fait que l'ensemble des sous-problèmes résolus est défini par une décomposition arborescente enracinée (C,T) : RDS-BTD résout $|C|$ sous-problème ordonnés par un parcours en profondeur de T, démarrant aux feuilles jusqu'à la racine $P^{\mathrm{RDS}}_1 = P_1$.

Nous définissons P^{RDS}_e comme le sous-problème défini par les variables *propres* de C_e ainsi que par celles de tous ses clusters descendants dans T et par les fonctions de coût impliquant *uniquement* des variables propres de ces clusters. P^{RDS}_e n'a aucun fonction de coût impliquant une variable de S_e, le séparateur avec son cluster père, et ainsi son coût optimal est un minorant de P_e conditionné par *n'importe quel* affectation de S_e. Ce coût optimal sera donc noté $LB^{\mathrm{RDS}}_{P_e}$

Chaque sous-problème P^{RDS}_e est résolu par BTD et non par DFBB. Ceci permet d'exploiter la décomposition et la mémorisation de minorants effectuée dans BTD, offrant ainsi une complexité asymptotique améliorée. Le minorant exploité peut alors s'appuyer sur les minorants $LB^{\mathrm{RDS}}_{P_e}$ déjà calculés sur des clusters déjà explorés, sur les minorants fournis par la cohérence locale et sur les minorants enregistrés par BTD[1]. Le minorant correspondant à l'affectation courante A est maintenant défini de façon récursive par $lb(P_e/A) = w^e_\varnothing \oplus \bigoplus_{C_f \in Fils(C_e)} \max(lb(P_f/A), LB_{P_f/A_f}, LB^{\mathrm{RDS}}_{P_f})$, qui est évidemment plus fort.

Dans BTD, la mémorisation n'est effectuée que lorsque les séparateurs sont complètement affectés alors que P^{RDS}_e ne contient pas les variables du séparateur S_e. Nous affectons donc S_e avant de résoudre P^{RDS}_e en utilisant les valeurs totalement supportées (full support) fournies par EDAC [10][2] de chaque variable comme des valeurs temporaires utilisées uniquement pour la mémorisation. Une approche alternative consisterait à mémoriser les minorants d'affectations partielles mais ceci nécessiterait un mécanisme de mémorisation complexe, à considérer dans des travaux futurs.

L'avantage de l'utilisation de BTD réside aussi dans le fait que les minorants enregistrés peuvent aussi être exploités dans les itérations suivantes de RDS-

[1] En fait, du fait que le filtrage par cohérence locale peut déplacer des coûts entre les clusters, le minorant $LB^{\mathrm{RDS}}_{P_f}$ doit être ajusté. Ceci est rendu possible par l'utilisation de la structure de données ΔW introduite dans [9] en soustrayant $\bigoplus_{i \in S_f} \max_{a \in D_i} \Delta W^f_i(a))$.

[2] Une valeur $a \in D_i$ totalement supportée vérifie $w_i(a) = 0$ et $\forall w_S \in W$ *with* $i \in S, \exists t \in \ell(S)$ *with* $t[i] = a$ tel que $w_S(t) = 0$.

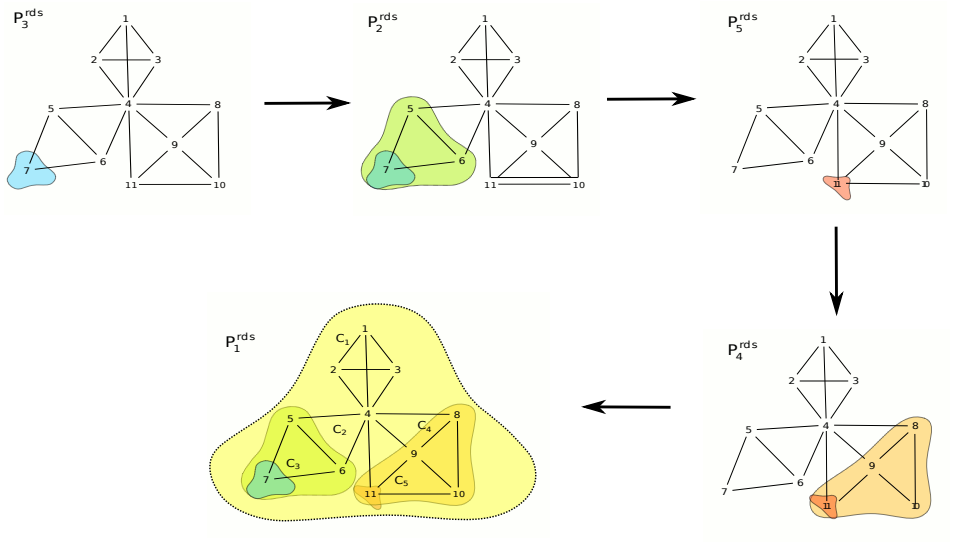

FIG. 2 – Résolution récursive des sous-problèmes relaxés issus de la Figure 1 par RDS-BTD.

BTD. Cependant, un optimum mémorisé par BTD pour un sous-problème P_f alors que P_e^{RDS} était résolu n'est plus nécessairement un optimum de $P_{Père(e)}^{\text{RDS}}$ si une fonction de coût existe entre P_f et les variables de $C_{Père(e)}$. De ce fait, à chaque itération de RDS-BTD, après la résolution de P_e^{RDS}, tous les drapeaux Opt_{P_f/A_f} tels que $S_f \cap S_e \neq \varnothing$ sont réinitialisés (line 4).

Exemple 3 *Considérons l'exemple 1, RDS-BTD traite successivement les sous-problèmes* P_3^{RDS}, P_2^{RDS}, P_5^{RDS}, P_4^{RDS} *et* $P_1^{\text{RDS}} = P_1$. P_3^{RDS} *contient seulement la variable* $\{7\}$ *et la fonction de coût* $\{w_7\}$. *Avant de résoudre* P_3^{RDS}, *RDS-BTD affecte les variables* $\{5, 6\}$ *du séparateur* S_3 *à leur valeurs totalement supportées* ($\{(5, a), (6, a)\}$ *ici). Quand* P_2^{RDS} *est résolu, l'optimum de* $P_3/\{(5, a), (6, a)\}$, *qui est égal à zéro car* $w_{5,6}$ *n'appartient pas à* P_3, *est enregistré et peut être réutilisé lors de la résolution de* P_1. *Quand* P_4^{RDS} *est résolu, l'optimum de* $P_5/\{(4, a), (9, a), (10, a)\}$, *qui est également nul, est enregistré. Dans ce cas, du fait que la variable* 4 *appartient à* $S_5 \cap S_4$ *et puisque* P_4^{RDS} *ne contient pas* $w_{4,11}$, *cet optimum enregistré n'est en fait qu'un minorant pour les itérations suivantes de RDS-BTD. De ce fait, nous fixons les booléens* $Opt_{P_5/\{(4,a),(9,a),(10,a)\}}$ *à faux avant de résoudre* P_1.

Les optima obtenus sont $LB_{P_3}^{\text{RDS}} = LB_{P_5}^{\text{RDS}} = 0$, $LB_{P_2}^{\text{RDS}} = LB_{P_4}^{\text{RDS}} = 1$ *et* $LB_{P_1}^{\text{RDS}} = 5$, *l'optimum de* P_1.

Dans cet exemple simple, pour une affectation initiale vide $A = \varnothing$, *on obtient un minorant* $lb(P_1/\varnothing) = LB_{P_2^{\text{RDS}}} \oplus LB_{P_4^{\text{RDS}}} = 2$ *au lieu de* 0 *pour BTD (en faisant l'hypothèse que la cohérence locale EDAC est utilisée en pré-traitement et qu'aucun majorant initial n'est fourni).*

Le pseudo-code de l'algorithme RDS-BTD est présenté ci-aprés comme l'algorithme 1. L'algorithme BTD utilisé est le même que dans [9] si ce n'est pour le fait qu'il exploite le minorant RDS amélioré défini ci-dessus. Nous faisons l'hypothèse que la résolution de P_e^{RDS} avec une affectation initiale A du séparateur S_e et un majorant initial cub_e obtenu par BTD est efféctuéé par l'appel à $\mathsf{BTD}(P_e^{\text{RDS}}, A, V_e, cub_e)$.

RDS-BTD appelle BTD pour résoudre chaque sous-problème P_e^{RDS} (ligne 3). Un majorant initial pour P_e^{RDS} est déduit du majorant global et des minorants RDS disponibles (ligne 1). Comme expliqué ci-dessus, les variables de S_e sont affectées à leurs valeurs totalement supportées à la ligne 2 . L'appel initial est simplement $\mathsf{RDS\text{-}BTD}(P)$. Il suppose que le problème $P_1^{\text{RDS}} = P$ est déjà localement cohérent et retourne son coût optimal.

Notez que dès qu'une solution de P_e^{RDS} ayant le même coût optimal que $lb(P_e^{\text{RDS}}/\varnothing) = \bigoplus_{C_f \in Fils(C_e)} LB_{P_f^{\text{RDS}}}$ est trouvée, la recherche s"arrête.

Les complexités temporelles et spatiales de RDS-BTD sont celles de BTD. Notez que sans mémorisation, en utilisant uniquement un filtrage par cohérence de nœud et une décomposition induite par un pseudo-arbre (avec un cluster pour chaque variable et un ordre statique des variables), RDS-BTD est équivalent à l'algorithme Pseudo-Tree RDS [14]. Si

Algorithm 1: RDS-BTD algorithme

Function RDS-BTD(P_e^{RDS}) : $[0, +\infty]$
  foreach $C_f \in Sons(C_e)$ **do** RDS-BTD(P_f^{RDS}) ;
1 $cub_e := cub_1 - lb(P/\varnothing) + lb(P_e^{\text{RDS}}/\varnothing)$;
2 Soit A l'affectation de S_e à des valeurs totalement supportées;
3 $LB_{P_e}^{\text{RDS}} :=$ BTD(P_e^{RDS}, A, V_e, cub_e) ;
  foreach C_f descendant de C_e *t.q.* $S_f \cap S_e \neq \varnothing$ **do**
4 Fixer à *faux* les $Opt_{P_f/A}$ enregistrés, pour tout $A \in \ell(S_f)$;
  return $LB_{P_e}^{\text{RDS}}$;

de plus, nous restreignons l'algorithme à l'utilisation d'une décomposition arborescente spécifique (C, T) telle que $|C| = n, \forall e \in [1, n], C_e = \{1, \ldots, e\}$, et $\forall e \in [2, n], Père(C_e) = C_{e-1}$, alors BTD-RDS se réduit à RDS.

5 Résultats expérimentaux

Nous avons codé DFBB, BTD, RDS-BTD, et RDS dans `toulbar2` un solveur C++ de CSP pondérés[3]. A l'intérieur des clusters, l'heuristique dynamique de choix de variable *min domain / max degree* est utilisée (par BTD et RDS-BTD) et par DFBB, les ex-aequo sont classés par les coûts unaires maximum. Cette heuristique est modifiée par une heuristique de type "conflict back-jumping" comme cela est proposé dans [16]. Durant la recherche, un niveau de cohérence locale EDAC [10] est maintenu à chaque étape. RDS s'appuie sur la cohérence de nœud [15] seulement. Les décompositions arborescentes sont construites par l'heuristique "Maximum Cardinality Search" (MCS), en utilisant le plus grand cluster comme racine (excepté pour le problème `scen07`). À partir de la décomposition arborescente fournie par MCS, nous avons dérivé des décompositions arborescentes alternatives définies par une taille de séparateur maximum s_{max} comme cela est proposé dans [13] : en partant des feuilles de la décomposition, nous fusionnons un cluster avec son père si la taille du séparateur dépasse s_{max}. Un ordre des variables compatible avec la décomposition arborescente enracinée exploitée est utilisé pour l'établissement de la cohérence locale DAC [6] ainsi que par RDS.

Les minorants enregistrés (et si disponibles, fournis par RDS) sont exploités par le filtrage par cohérence locale dés que le séparateur correspondant est totalement affecté. Si le minorant enregistré est optimal ou supérieur au minorant fourni par EDAC, le sous-problème (P_e/A_e) correspondant est déconnecté du filtrage par cohérence locale et la différence positive entre les minorants est ajoutée au minorant du cluster père ($w_{\varnothing}^{Père(C_e)}$), conduisant à de nouveaux effacements de valeurs par cohérence de nœud.

Toutes les méthodes s'appuient sur un schéma de branchement binaire. Si $d > 10$, le domaine *ordonné* est coupé en deux parties (autour de la valeur centrale) sinon, la variable est affectée à son support complet ou cette valeur est retirée du domaine. Dans les deux cas, la branche contenant la valeur support complet est explorée en premier, excepté pour RDS et les méthodes dérivées de BTD qui choisissent en priorité (si elle existe) la branche qui contient la valeur de la dernière solution trouvée. Sauf mention précise, aucun majorant initial n'est fourni. Les temps CPU présentés correspondent au temps nécessaire pour trouver une solution optimale et prouver son optimalité.

5.1 Affectation de fréquences

Parmi les différentes instances du CELAR [2] qui peuvent être décrites comme un CSP pondéré binaire, nous avons sélectionné deux instances difficiles : `scen06` and `scen07`. L'instance `scen07` a été réduite par différentes règles de prétraitement (filtrage des domaines par singleton EDAC, par les règles de dominance proposées par Koster [8], élimination des variables de degré faible) conduisant à une instance où $n = 162$, $d = 44$ et $e = 764$. La décomposition arborescente utilisée a un $s_{max} = 3$ et une largeur d'arbre de 53. Un majorant initial (354008) a été fourni à toutes les méthodes. Ce majorant et la solution correspondante ont été trouvés par un algorithme de type DFBB utilisant un mécanisme de type "Limited Discrepancy Search". Cette solution est également utilisée par RDS-BTD comme choix de valeur initial (les performances de DFBB et BTD sont dégradées par ce choix). Le cluster racine a été choisi sur la base du "First Fail Principle" (cluster de coût optimal maximum). BTD et RDS-BTD trouvent l'optimum de 343592 et prouvent l'optimalité en respectivement 6 et 4.5 jours (amélioration de 26% pour RDS-BTD) sur un processeur à 2.6 GHz équipé de 32GB de mémoire. BTD a enregistré 90528 minorants (20% de l'espace des affectations de séparateurs) comparés aux 29453 (6.4%) mémorisés par RDS-BTD. DFBB n'a pas terminé en 50 jours. C'est la première résolution avec preuve d'optimalité de ce problème ouvert depuis plus de 10 ans.

Nous avons aussi résolu l'instance `scen06` (100 variables avec une largeur d'arbre de 11) sans aucun prétraitement ni majorant initial. BTD et RDS-BTD ont pris 221 et 316 secondes respectivement pour trouver l'optimum et prouver l'optimalité. Ils ont respective-

[3] https ://mulcyber.toulouse.inra.fr/projects/toulbar2 version 0.8.

ment enregistré 5633 et 7145 minorants malgré le fait que la taille des séparateurs n'ait pas été restreinte (on observe un $s_{max} = 8$). DFBB a pris 2588 secondes et RDS n'a pas terminé en 10 heures.

5.2 Sélection de Tag SNP

Ce problème apparaît en génétique et analyse du polymorphisme. Les SNP (ou Single Nucleotide Polymorphism, prononcé snip) sont des variations ponctuelles dans le génome d'individus d'une même espèce. Il se caractérise par une altération d'un seul nucléotide (`A`,`T`,`C`,or `G`) dans la séquence. Par exemple, un SNP peut changer la séquence D'ADN `AAGGCTAA` en `ATGGCTAA`. Pour qu'une variation soit considérée comme un SNP, elle doit apparaître dans au moins 1% de la population considérée. Dans les trois milliards de nucléotides que compte le génome humain, on dénombre plusieurs millions de SNP. Ils expliquent jusqu'à 90% de toutes les variations génétiques humaines, notamment une partie du risque héritable de maladies communes ainsi que la susceptibilité de réponses aux pathogènes, produits chimiques, médicaments vaccins et autres agents.

Le problème TagSNP est une forme de compression d'information avec perte consistant à sélectionner un sous-ensemble de SNP tel que les SNP sélectionnés (appelés tag SNP) capturent l'essentiel de l'information génétique. Le but est de capturer un ensemble de petite taille le plus informatif possible afin de rendre possible le criblage et l'analyse statistique d'une population de grande taille [12].

La mesure de corrélation r^2 entre une paire de SNP peut dans un premier temps être déterminée sur une petite population. Un tag SNP est considéré comme "représentatif" d'un autre SNP si les deux SNP sont suffisamment corrélés. Le problème de sélection de tag SNP le plus simple consiste à sélectionner un nombre minimum de tag SNP de façon à ce que tous les SNP soient représentés. Ceci est capturé par le fait que la mesure r^2 entre deux SNP est supérieure à un seuil θ (souvent fixé à $\theta = 0.8$ [4]). Nous considérons donc un graphe dans lequel chaque sommet est un SNP et où les arêtes sont pondérées par la mesure r^2 entre les SNP des sommets. Les arêtes dont le seuil est plus petit que θ sont supprimées. Le graphe filtré ainsi obtenu peut avoir plusieurs composantes connexes. Le problème TagSNP se réduit alors à un problème de couverture d'ensemble (set covering, NP-dur) sur chacune des composantes. Cette formulation simple du problème a été étudiée avec de bons résultats dans [1] via le compilateur `c2d` .

En pratique, le nombre de solutions optimales peut être très important et les outils spécialisés du domaines tel que FESTA [18], (en s'appuyant sur deux algorithmes incomplets), optimisent, en plus du nombre de tagsnp selection, des critères secondaires : Entre deux tag SNP, une mesure r^2 faible est préférée afin de maximiser la dispersion des tag SNP. Entre un non tag SNP et un de ses représentants, une mesure r^2 élevée est préférée afin de maximiser la représentativité de la sélection de tag SNP effectuée.

Pour un graphe connexe donné $G = (V, E)$, nous construisons un WCSP binaire avec des coûts entiers capturant le problème TagSNP avec les critères secondaires précédents. Pour chaque SNP i, deux variables i_s et i_r sont introduites. i_s est une variable booléenne indiquant si le SNP est sélectionné comme tag SNP ou non. Le domaine de i_r est formé par l'ensemble des voisins de i complété avec i lui-même. Il indique le tag SNP qui est chargé de représenter i. Pour un SNP i, des fonctions de coût dures (avec des coûts nuls ou infinis) établissent le fait que $i_s \Rightarrow (i_r = i)$. Des fonctions de coût dures similaires établissent $(i_r = j) \Rightarrow j_s$ avec les SNP j voisins dans G. Une fonction de coût unaire sur chaque variable i_s produit un coût élémentaire U si la variable est à *vrai*. Le WCSP résultant capture déjà le problème de couverture d'ensembles pur défini par le problème TagSNP.

Pour prendre en compte le critère de représentativité, une fonction de coût unaire est associée avec chaque variable i_r . Si $i_r \neq i$, elle génère un coût non nul égal à $\lfloor 100.\frac{1-r^2_{i,i_r}}{1-\theta} \rfloor$. Pour capturer la dispersion entre tag SNPs i et j, une fonction de coût binaire entre les booléens i_s et j_s est introduite. Elle produit un coût de $\lfloor 100.\frac{r^2_{ij}-\theta}{1-\theta} \rfloor$ lorsque $i_s = j_s = vrai$. Le WCSP ainsi obtenu capture simultanément les critères de dispersion et de représentativité. Afin de garder à ces critères leur caractère secondaire, nous utilisons simplement une valeur de coût U (le coût de sélection d'un SNP) suffisamment grande. En pratique, supérieur à la somme des critère secondaire et primaire dans l'instance considérée.

Ce problème est similaire à un problème de couverture d'ensembles (set covering) mais avec des coûts additionnels binaires (quadratiques). Ces critères sont ignorés par [1]. Dans cet article, `c2d` donne une représentation compacte de l'ensemble des solutions du problème de couverture simple, mais le nombre de solutions optimales est si grand (typiquement supérieur à des milliards) que l'application des critères secondaires à des solutions générées par `c2d` serait trop coûteuse. Comme le mentionnent les auteurs dans leur conclusion, une compilation directe des critères secondaire dans le d-DNNF ne semble pas immédiate. Elle nécessiterait une reformulation du problème sous forme MaxSAT.

Les instances considérées dérivent de données obtenues sur chromosome humain numéro 1, aimable-

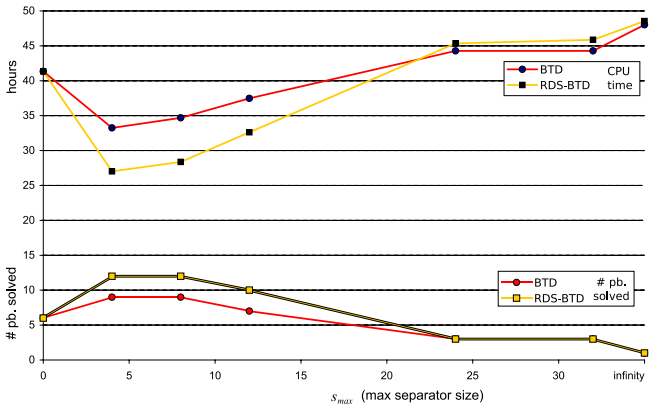

FIG. 3 – Temps de résolution global et nombre de problèmes résolus en faisant varier la taille maximale du séparateur s_{max}.

ment fournies par Steve Qin [18]. Deux valeurs de seuil, $\theta = 0.8$ et 0.5 ont été essayées. Pour $\theta = 0.8$, une valeur usuelle dans le domaine, 43251 composantes connexes ont été identifiées parmi lesquelles nous avons sélectionné les 82 plus grandes. Ces problèmes, qui incluent de 33 à 464 SNP, définissent des WCSP avec des tailles de domaine entre 15 et 224. Ces instances sont relativement faciles. Leur résolution à l'optimalité sélectionne 359 tag SNPs en 2h37' au lieu des 487 SNP sélectionnés en 3' par FESTA (en mode greedy), soit 21% d'amélioration. En mode hybride, FESTA sélectionne 370 tag SNP en 39h17' soit un modeste gain de 3% dans notre cas, mais avec une amélioration d'un facteur 15 en temps.

Afin d'obtenir des problèmes plus difficiles, nous avons abaissé le seuil θ à 0.5. Ceci défini $19,750$ composantes connexes parmi lesquelles 516non triviales sont résolues par FESTA par approches incomplètes. Nous avons sélectionnées les 25 plus grandes. Ces instances contiennent de 171 à 777 SNP et ont une densité d'arête qui varie entre 6% et 37%. Elles définissent des instances WCSP dont la taille des domaines varie entre 30 et 266 et contenant de 8000 à $250,000$ fonctions de coûts. La décomposabilité de ces problèmes, estimée par le ratio entre la largeur d'arbre d'une décomposition fournie par MCS ($s_{max} = +\infty$) et le nombre de variables, varie de 14% à 23%.

Tous les problèmes ont été traités avec un majorant initial fourmi par FESTA (mode greedy). L'éxperimentation a été réalisé sur machine équipée de CPU à 2.8 Ghz et 32 GB de RAM. Pour mettre en valeur l'importance de l'utilisation d'une taille de séparateurs bornée (via s_{max}), nous avons considéré des valeurs de s_{max} allant de 0 (DFBB), 4, 8, 12, 24, 32 à $+\infty$ pour les algorithmes BTD et RDS-BTD. Nous présentons le nombre de problèmes résolus dans une limite de temps de deux heures ainsi que le temps CPU global utilisé (une instance non résolue comptant pour deux heures). RDS présentant de mauvaises performances, les résultats correspondants ne sont pas présentés.

En utilisant $s_{max} = 4$, notre implémentation améliore le taux de compression de FESTA-greedy par un rapport de 15% (en sélectionnant 2952 tag SNP au lieu de 3477 pour les $516-13$ instances résolues). Notez que la différence en temps CPU entre BTD et RDS-BTD augmenterait si une plus grande limite de temps était utilisée. D'un point de vue pratique, le critère utilisé dans le problème TagSNP pourrait être raffinée encore en incluant des informations variées : informations d'annotation de séquences (*e.g.* préférant les tag SNP apparaissant dans les gènes), mesure entre triplets de SNP comme proposé dans [1] (SNP couverts par des paires de tag SNP). Les bonnes performances de RDS-BTD pourraient permettre de résoudre ces problèmes avec un seuil réaliste de $\theta = 0.8$.

La Figure 4 représente, pour chaque instance ($\theta = 0.5$) résolue au moins une fois lors des expérimentations précédentes, l'évolution individuelle du ratio largeur d'arbre de la décomposition sur taille du problème en fonction de s_{max}. Le succès de la résolution (preuve d'optimalité obtenue dans le temps imparti) est matérialisé par un trait continu, à l'inverse les échecs sont représentés par des pointillés. L'unique résolution de l'instance "17034" (s_{max}=4), est représentée par un carré gris.

Ainsi, on retrouve individuellement les 12 instances résolues pour $s_{max} = \{4, 8\}$, mais il est intéréssant de noter que la nature des instances clôturées sont légèrement différentes. L'instance "17034" est résolue uniquement à s_{max}=4, ce qui est compensé par l'instance "14359" , qui est résolue sur l'intervalle 8 et 12. Au total, 13 instances sont clôturées sur l'ensemble des

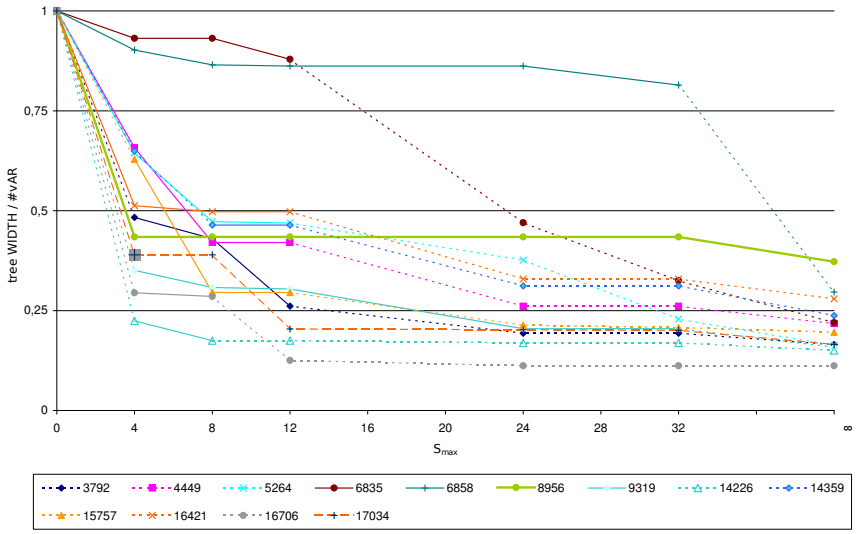

FIG. 4 – Evolution de la largeur d'arbre normalisée par le nombre de variables en fonction de la taille maximale des séparateurs s_{max}.

expérimentations. Une diminution importante dans la largeur d'arbre de décomposition est souvent corrélée au succès de la résolution du problème.

Pour la majorité des instances étudiées, reflétant la structure des problémes, les courbes montrent une zone de décroissance importante pour $s_{max} \in [0, 8]$, qui est sans doute à l'origine des bons résultats obtenus. La diminution de la largeur d'arbre est ensuite plus faible. Elle atteint progressivement ($s_{max} > 12$), un plateau de pente faible, qui tends asymptotiquement vers le ratio associé à la décomposition complète du problème. Cette zone des courbes est souvent associée à un échec dans les expérimentations. Ainsi par exemple, l'instance 15757 est résolue pour des valeurs de s_{max} de 4, 8, 12 et 24 avec respectivement des temps de 514, 7, 299 et 3810 secondes. Sur le plan de la décomposition, le ratio de largeur d'arbre passe de $0,63$ pour s_{max}=4 à $0,30$ pour $s_{max} = 8$ et $s_{max} = 12$ pour enfin avoisiner $0,2$ à $s_{max} = 24$. La résolution de l'instance est optimale à $s_{max} = 8$. Le cas de cette instance n'est pas isolé. Comme le montre la Figure 3, la plupart des instances résolues présentent une valeur s_{max} *optimale* pour la résolution. Cette situation correspond, probablement, à un bon compromis entre gains issus de la décomposition et pertes liées à une diminution dans la liberté de l'ordre de choix des variables.

6 Conclusion

L'exploitation pratique de décompositions arborescentes pour résoudre des problèmes d'optimisation combinatoire qui ont une structure n'est pas toujours immédiate. Les problèmes d'optimisation définissent de façon explicite un critère global qui doit être optimisé sur l'ensemble de l'instance. En résolvant des sous-problèmes conditionnellement indépendants, les algorithmes exploitant une décomposition arborescente des problèmes perdent la vision globale sur le critère en exploitant des bornes locales assez faibles. En fournissant des minorants forts pour chacun des sous-problèmes, l'approche des poupées russes permet d'injecter une information plus globale dans chaque résolution de sous-problème au travers d'un majorant initial renforcé, évitant ainsi nombre de résolution inutiles et menant à une efficacité accrue.

Au delà, nos expérimentations montrent que, même sur des problèmes ayant une structure forte, il est souvent profitable et parfois indispensable de restreindre la décomposition en bornant la taille des séparateurs. La théorie nous indique que la taille des séparateurs influe sur la complexité spatiale d'algorithmes tels que BTD ou RDS-BTD, mais même lorsque la taille des séparateurs n'est pas restreinte, aucun des instances

considérées dans nos tests n'a pu épuiser la mémoire disponible. En pratique, l'amélioration en efficacité est d'abord explicable par la liberté supplémentaire dans l'ordonnancement des variables rendue possible par la fusion de clusters. Cette observation est cohérente avec les conclusions de [13]. On notera aussi que le nombre d'instanciations possibles d'un séparateur croît exponentiellement avec sa taille. Plus le séparateur est grand, et plus la probabilité qu'un minorant mémorisé serve à nouveau devient faible : la mémorisation est sans doute surtout utile dans les séparateurs de petite taille.

Concernant notre algorithme, la synergie entre RDS et BTD pourrait être améliorée en autorisant BTD à mémoriser des minorants associés à des instanciations partielles des séparateurs. Cela permettrait de réduire encore les recherches redondantes entre itérations successives de RDS-BTD et serait également profitable dans une stratégie d'approfondissement itératif (iterative deepening) comme cela a été proposé dans [3].

Dans nos tests, le seuil s_{max} a été appliqué de façon uniforme à toutes les instances. Il reste à étudier à quel point un ajustement de ce seuil par instance, en prenant en compte la taille des séparateurs existants dans le problème et les variations de largeur d'arbre qu'ils peuvent induire, pourrait améliorer les performances de l'algorithme.

Acknowledgments Ce travail a été partiellement financé par l'*Agence Nationale de la Recherche* (projet `STALDECOPT`).

Références

[1] A. Choi, N. Zaitlen, B. Han, K. Pipatsrisawat, A. Darwiche, and E. Eskin. Efficient Genome Wide Tagging by Reduction to SAT. In *Proc. of WABI-08*, volume 5251 of *LNCS*, pages 135–147, 2008.

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

[3] B. Cabon, S. de Givry, and G. Verfaillie. Anytime Lower Bounds for Constraint Optimization Problems. In *Proc. of CP-98*, pages 117–131, Pisa, Italy, 1998.

[4] C. S. Carlson, M. A. Eberle, M. J. Rieder, Q. Yi, L. Kruglyak, and D. A. Nickerson. Selecting a maximally informative set of single-nucleotide polymorphisms for association analyses using linkage disequilibrium. *Am. J. Hum. Genet.*, 74(1) :106–120, 2004.

[5] M. Cooper, S. de Givry, and T. Schiex. Optimal soft arc consistency. In *Proc. of IJCAI-07*, pages 68–73, Hyderabad, India, 2007.

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

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

[8] S. de Givry. Singleton consistency and dominance for weighted CSP. In *Proc. of Soft Constraints workshop*, 2004.

[9] S. de Givry, T. Schiex, and G. Verfaillie. Exploiting Tree Decomposition and Soft Local Consistency in Weighted CSP. In *Proc. of AAAI-06*, Boston, MA, 2006.

[10] 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.

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

[12] J.N. Hirschhorn and M.J. Daly. Genome-wide association studies for common diseases and complex traits. *Nature Reviews Genetics*, 6(2) :95–108, 2005.

[13] P. Jégou, S. N. Ndiaye, and C. Terrioux. Dynamic management of heuristics for solving structured CSPs. In *Proc. of CP-07*, pages 364–378, Providence, USA, 2007.

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

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

[16] C. Lecoutre, L. Sais, S. Tabary, and V. Vidal. Last conflict based reasoning. In *Proc. of ECAI-2006*, pages 133–137, Trento, Italy, 2006.

[17] R. Marinescu and R. Dechter. Memory intensive branch-and-bound search for graphical models. In *Proc. of AAAI-06*, Boston, MA, 2006.

[18] Z. S. Qin, S. Gopalakrishnan, and G. R. Abecasis. An efficient comprehensive search algorithm for tagsnp selection using linkage disequilibrium criteria. *Bioinformatics*, 22(2) :220–225, 2006.

[19] M. Sanchez, S. de Givry, and T. Schiex. Mendelian error detection in complex pedigrees using weighted constraint satisfaction techniques. *Constraints*, 13(1) :130–154, 2008.

[20] T. Schiex. Arc consistency for soft constraints. In *Proc. of CP-2000*, pages 411–424, Singapore, 2000.

[21] T. Schiex, H. Fargier, and G. Verfaillie. Valued constraint satisfaction problems : hard and easy problems. In *Proc. of the 14^{th} IJCAI*, pages 631–637, Montréal, Canada, 1995.

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

[23] G. Verfaillie, M. Lemaître, and T. Schiex. Russian Doll Search for Solving Constraint Optimization Problems. In *Proc. of AAAI-96*, pages 181–187, Portland, OR, 1996.