Ecole Doctorale STIM : Sciences
et Technologies de l'Information
et Mathématiques

Spécialité : Informatique
Laboratoire : LINA
Equipe : TASC

Une approche basée sur les contraintes pour résoudre le problème d'arbre recouvrant de coût minimum avec contraintes de degré

Jean-Guillaume Fages
Mél : jean-guillaume.fages@mines-nantes.fr

Résumé : De nombreuses applications industrielles requièrent la résolution de problèmes combinatoires issus de la Théorie des Graphes. Cet article présente un modèle de Programmation par Contraintes pour résoudre le problème de recherche d'un Arbre Recouvrant de Poids Minimum sous Contraintes de Degré (ARPMCD). Cette approche générale, basée sur une relaxation Lagrangienne, est compétitive avec la méthode dédiée de Da Cunha et Lucena qui fait état de l'art. De plus, elle améliore la résolution des instances à coûts aléatoires d'un à deux ordres de grandeur.

Mots clés : *Optimisation, Théorie des graphes, Contraintes*

1 Introduction

Etant donné un graphe non orienté $G = (V, E)$ muni d'une fonction de coût $f : E \to \mathbb{Z}$ et d'une fonction $d_{max} : V \to \mathbb{N}$, le problème de recherche d'un Arbre Recouvrant de Poids Minimum sous Contraintes de Degré (ARPMCD) consiste à trouver un Arbre Recouvrant de Poids Minimum (ARMP) dans G, tel que tout nœud $v \in V$ a, au plus, $d_{max}(v)$ voisins. Ce problème NP-difficile [1] peut survenir dans de nombreuses applications industrielles impliquant des réseaux. Les plus communes sont les application de conception de réseaux de télécommunication où des modules doivent être connecté ensembles, chacun d'eux ayant un nombre limité de ports de connexion disponibles. De telles restrictions peuvent être dues aux composants matériels utilisés mais aussi à des mesures de sécurité sur le réseau.

Plusieurs méthodes, exactes et heuristiques, ont été proposées pour résoudre le problème de l'ARPMCD [2, 3, 4, 5, 6, 7]. Actuellement, la meilleure approche de l'état de l'art est la méthode « *Relax and Cut and Branch and Cut* » (`RCBC`) [5], qui implique une relaxation Lagrangienne renforcée par l'ajout de coupes (de type « *Blossom* ») et d'une procédure de recherche locale qui démarre à chaud un algorithme de « *Branch and Cut* ». Il est important de remarquer que le problème d'ARPMCD n'est en général qu'un sous-problème des préoccupations industrielles, qui peuvent inclure de nombreuses contraintes annexes qu'il serait bon d'intégrer au modèle. Cependant, la recherche locale, proposée par [5], n'est pas effectuée par un solveur générique mais par un algorithme dédié intégré au cœur même de la relaxation Lagrangienne. D'une redoutable efficacité, une telle approche détériore les capacités d'évolutions de leur modèle. A l'opposé, la Programmation par Contraintes est une technologie qui permet une prise en compte aisée des spécificités de ce type d'applications [8].

Ce papier introduit une approche en Programmation par Contraintes qui permet de fait une intégration facile des contraintes annexes, pour résoudre le problème d'ARPMCD. La force du modèle proposé repose sur la complémentarité de ses composants : il inclut une optimisation du bas vers le haut et un propagateur de relaxation Lagrangienne, dont le filtrage est directement relié à la borne supérieure de la variable objectif, ce qui forme une combinaison puissante. Ces raisonnements basés sur les coûts sont renforcés en considérant les propriétés structurelles du problème. Des propagateurs basés sur la structure du graphe réduisent l'espace de recherche alors que l'heuristique de branchement tend à créer de plus en plus de structure. Ce modèle améliore la résolution des instances dîtes aléatoire d'un à deux ordres de grandeur.

Cet article est organisé comme suit : Tout d'abord, la section 2 introduit le modèle de Programmation par Contraintes. Ensuite, la section 2 le compare à l'approche de l'état de l'art. Finalement, une brève conclusion est donnée en section 4.

2 Le modèle

Le problème d'ARPMCD peut être formalisé comme suit :

$$
\begin{array}{lll}
\textit{minimize} & \sum_{(i,j)\in G'} f(i,j) & (1)\\
\textit{subject to :} & arbre(G') & (2)\\
& |\{(i,j) \in G'\}| \leq d_{max}(i) \quad ,\forall i \in V & (3)\\
& G' = (V, E' \subseteq E) & (4)
\end{array}
$$

Notre modèle implique deux variables : une variable entière, $z \in \mathbb{Z}$, représente la fonction objectif (1) ; une variable graphe [9, 10, 11], notée G', représente la solution du problème, *i.e.* un arbre recouvrant. La définition du domaine initial de G' (4) indique que le graphe solution doit contenir tous les nœuds de G et un sous ensemble de ses arêtes. Une contrainte globale de graphe $arbre$ (2) assure que G' forme bien un arbre (nécessairement recouvrant puisque tous les nœuds sont obligatoires). Cette contrainte empêche G' de contenir des cycles avec un propageur similaire à celui de la contrainte $noCycle$ [12] et assure la connexité de G' grâce à un propagateur, basé sur l'algorithme de Tarjan [13], qui détecte et force les isthmes. Les contraintes de degré (3) sont prises en compte par une simple contrainte de cardinalité. Finalement, le lien entre G' et z est fait via une contrainte de type $scalaire$ (1).

2.1 Raisonnement sur les faibles degrés

En règle générale, plus le degré maximum est faible, plus l'instance est difficile [2, 5, 6]. Cependant, lorsque le degré maximum de quelques nœuds vaut un, cela apporte une certaine structure au graphe, qui peut alors être exploitée dans le cadre de la Programmation par Contraintes. On remarque ainsi que, si $|V| > 2$, deux nœuds dont le degré maximum vaut 1 ne peuvent pas être connectés par une arête, car la contrainte de degré les déconnecterait du reste du graphe (voir figure 1(b)). Cette règle peut être généralisée aux autres nœuds du graphe : considérons qu'à chaque fois qu'une arête $(i,j) \in G'$, $d_{max}(i) = 1$ et $d_{max}(j) \neq 1$, est forcée, le nœud i (alors saturé) est retiré du graphe et le degré maximum de j est décrémenté de 1. Alors, s'il reste encore au moins trois nœuds dans le graphe, dès que $d_{max}(j)$ devient égal à 1, on peut filtrer toutes les arêtes $(j,k) \in G'$ telles que $d_{max}(k) = 1$. La figure 1 illustre cette règle de filtrage : en appliquant cette procédure au graphe de la figure 1(c), celui-ci devient isomorphe au graphe de la figure 1(b). On en déduit que l'arête (D,E) est impossible. Nous ajoutons donc à notre modèle un propagateur qui applique ce raisonnement pour filtrer des arêtes impossibles.

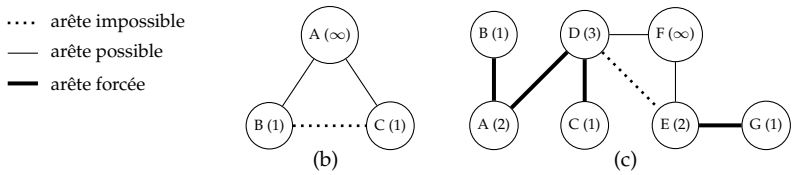


FIGURE 1 – Observation structurelle sur les degrés maximum des nœuds (affichés entre parenthèses).

2.2 Relaxation Lagrangienne

Malheureusement, la contrainte $scalaire$ maintenant la variable objectif ne va ni donner une bonne borne inférieure pour z, ni déclencher beaucoup de filtrage sur G'. Nous pourrions améliorer ce modèle en y ajoutant la contrainte *arbre recouvrant pondéré* [14, 15], plus puissante, mais cela ne serait toujours pas suffisant pour espérer battre les meilleures approches actuelles. Pour cette raison, nous introduisons un nouveau propagateur, *redondant* au modèle d'un point vue sémantique, qui applique la relaxation Lagrangienne du problème d'ARPMCD. En effet, des travaux similaires [16, 17] sur le problème du Voyageur De Commerce, qui peut être vu comme un cas particulier du problème d'ARPMCD, ont montré que la relaxation Lagrangienne basée sur un recouvrement de graphe par un arbre est très efficace. Cette relaxation est un processus itératif dans lequel chaque itération consiste à résoudre le problème d'ARPM et mettre à jour la fonction de coût en fonction des violations des contraintes de degré. Chaque itération pouvant donner lieu au filtrage décrit en [14, 15]. De plus amples informations sur cette relaxation peuvent être trouvée dans [2, 4].

2.3 Branchement

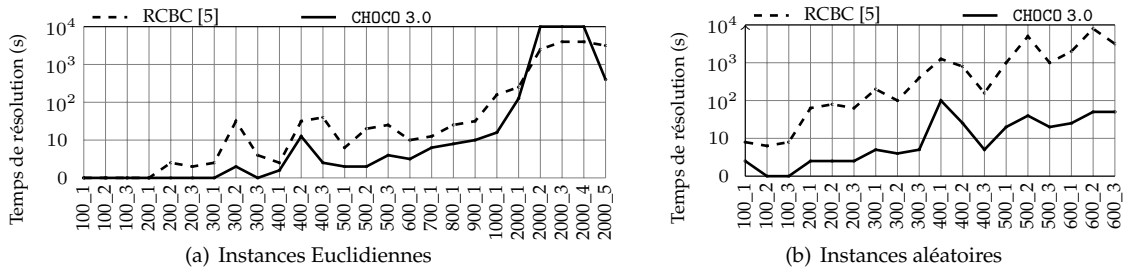
L'intervalle entre la première solution et l'optimum peut contenir un nombre considérable de solutions intermédiaires. Aussi, atteindre l'optimum en appliquant une optimisation par le haut (approche par défaut des solveurs de contraintes) risque de prendre un temps conséquent. Cependant, la relaxation Lagrangienne donne une borne inférieure de très bonne qualité (bien souvent à moins de 1% de la valeur optimale). Notre approche cherche donc une première solution, puis réinitialise la recherche en effectuant une optimisation par le bas : on cherche une solution de coût égal à la borne inférieure courante. Si aucune solution n'existe, on incrémente la borne inférieure. Il en découle que la deuxième solution trouvée est nécessairement optimale. Le gros de la recherche va se dérouler dans des régions non réalisables, aussi il est conseillé d'opter pour une approche de type « *fail-first* » [18] qui va tendre à provoquer les échecs. Pour cela, notre heuristique de branchement cherche à forcer les arêtes en dehors du support de la relaxation Lagrangienne. Cela fera travailler le propagateur Lagrangien et générera du filtrage. Nous renforçons enfin cette heuristique en cherchant à saturer les nœuds dont le degré maximum est à 1 aussi tôt que possible afin d'entretenir le propagateur défini en section 2.1.

3 Etude expérimentale

Cette section présente des expériences effectuées dans le but de positionner notre approche en Programmation par Contraintes par rapport à l'état de l'art. Elle démontre que notre modèle est compétitif avec la méthode dédiée de [5] et qu'elle offre de bonnes perspectives de passage à l'échelle sur de nombreuses instances.

Notre modèle a été implémenté en `Java`, avec le solveur `CHOCO 3.0` [1], et testé sur un Macbook pro sous OS X 10.7.2 avec un processeur Intel core i7 à 2.7 GHz et 8GB de mémoire DDR3. Une limite de 2 GB a été imposée sur la machine virtuelle Java (`JVM`). Malheureusement, le code de l'approche de l'état de l'art [5] n'est pas disponible, donc nous reportons directement les résultats de leur article. Leur code ayant été implémenté en `C`, l'écart de temps d'exécution des langages devrait compenser l'écart de performance des machines. Notre code est inclu dans la librairie d'exemples de `CHOCO 3.0`, il est donc accessible à tous.

Nous considérons deux jeux d'instances, nommés ANDINST et DR, qui font parti des plus difficiles de la littérature. Ils impliquent des graphes initiaux complets ayant de 100 à 2000 nœuds pour les instances ANDINST et de 100 à 600 nœuds pour les instances DR. Dans les instances de type ANDINST, chaque nœud est associé a un point du plan Euclidien, placé aléatoirement. La fonction de coût est la distance Euclidienne séparant les nœuds d'une arête. Le degré maximum de chaque nœud est ensuite choisi aléatoirement dans l'intervalle $[1, 4]$. Pour les instances de type DR, le coût de chaque arête a été choisi aléatoirement dans l'intervalle $[1, 1000]$, avec une distribution de probabilités uniforme, et le degré maximum de chaque nœud est choisi dans $[1, 3]$. Du point de vue des restrictions de degré, les instances DR sont donc plus difficiles.



(a) Instances Euclidiennes (b) Instances aléatoires

FIGURE 2 – Positionnement de l'approche par contraintes.

La figure 2(a) montre que l'approche par contraintes est compétitive avec l'état de l'art sur la plupart des instances Euclidiennes. Elle résout à l'optimum des instances impliquant jusqu'à deux millions d'arêtes, ce qui atteste d'une bonne aptitude au passage à l'échelle. Cependant, elle ne parvient pas à résoudre toutes ces instances dans le temps imparti (10^4 secondes). Il a été montré dans [5] qu'utiliser une relaxation linéaire augmentée de coupes de type « *Blossom* » apportait des gains significatifs sur les instances Euclidiennes. On pourrait donc s'attendre à améliorer l'approches par contraintes sur ce type d'instances, en ajoutant un propagateur qui applique cette relaxation.

1. `http://www.emn.fr/z-info/choco-solver/`

Concernant les instances aléatoires (figure 2(b)), notre approche améliore les résultats antérieurs d'un à deux ordres de grandeur. Cela démontre la pertinence d'un modèle basé sur les contraintes pour résoudre cette classe d'instance, qui était dîte difficile à résoudre [5]. La relaxation Lagrangienne, qui n'était utilisée que dans une phase amont, devrait plutôt être appliquée durant tout le processus de recherche, comme le fait notre approche. Aussi, l'utilisation de recherche locale et de programmation linéaire a plus tendance à ralentir qu'à aider la résolution de ces instances.

4 Conclusion

Nous avons proposé une première approche en Programmation par Contraintes pour résoudre le problème d'ARPM sous contraintes de degré. Ce modèle est plus flexible que la meilleure approche de l'état de l'art et il améliore significativement la résolution des instances aléatoires. Cette approche est donc appropriée pour concevoir une application industrielle amenée à évoluer.

Références

[1] Michael R. Garey and David S. Johnson. *COMPUTERS AND INTRACTABILITY*. W. H. Freeman and Company, victor klee edition, 1979.

[2] Rafael Andrade, Abilio Lucena, and Nelson Maculan. Using lagrangian dual information to generate degree constrained spanning trees. *Discrete Applied Mathematics*, 154(5) :703–717, 2006.

[3] Markus Behle, Michael Jünger, and Frauke Liers. A primal branch-and-cut algorithm for the degree-constrained minimum spanning tree problem. In *WEA*, pages 379–392, 2007.

[4] Alexandre Salles da Cunha and Abilio Lucena. Lower and upper bounds for the degree-constrained minimum spanning tree problem. *Networks*, 50(1) :55–66, 2007.

[5] Alexandre Salles da Cunha and Abilio Lucena. A hybrid relax-and-cut/branch and cut algorithm for the degree-constrained minimum spanning tree problem. Technical report, Universidade Federal do Rio de Janeiro, 2008.

[6] Mohan Krishnamoorthy, Andreas T. Ernst, and Yazid M. Sharaiha. Comparison of algorithms for the degree constrained minimum spanning tree. *J. Heuristics*, 7(6) :587–611, 2001.

[7] Celso C. Ribeiro and Maurício C. Souza. Variable neighborhood search for the degree-constrained minimum spanning tree problem. *Discrete Applied Mathematics*, 118(1-2) :43–54, 2002.

[8] Helmut Simonis. *Handbook of Constraint Programming*, *Constraint Applications in Networks*. Elsevier, victor klee edition, 2006.

[9] Grégoire Dooms, Yves Deville, and Pierre Dupont. CP(Graph) : Introducing a Graph Computation Domain in Constraint Programming. In *CP*, volume 3709, pages 211–225, 2005.

[10] Claude Le Pape, Laurent Perron, Jean-Charles Régin, and Paul Shaw. Robust and Parallel Solving of a Network Design Problem. In *CP*, volume 2470, pages 633–648, 2002.

[11] Jean-Charles. Régin. Tutorial : Modeling Problems in Constraint Programming. In *CP*, 2004.

[12] Yves Caseau and François Laburthe. Solving Small TSPs with Constraints. In *ICLP*, pages 316–330, 1997.

[13] Robert E. Tarjan. Depth-First Search and Linear Graph Algorithms. *SIAM Journal on Computing*, 1(2) :146–160, 1972.

[14] Jean-Charles Régin. Simpler and Incremental Consistency Checking and Arc Consistency Filtering Algorithms for the Weighted Spanning Tree Constraint. In *CPAIOR*, volume 5015, pages 233–247, 2008.

[15] Jean-Charles Régin, Louis-Martin Rousseau, Michel Rueher, and Willem Jan van Hoeve. The Weighted Spanning Tree Constraint Revisited. In *CPAIOR*, volume 6140, pages 287–291, 2010.

[16] Pascal Benchimol, Willem-Jan Van Hoeve, Jean-Charles Régin, Louis-Martin Rousseau, and Michel Rueher. Improved filtering for Weighted Circuit Constraints. *Constraints*, To appear.

[17] Jean-Guillaume Fages and Xavier Lorca. Improving the Asymmetric TSP by Considering Graph Structure. Technical report, Ecole des Mines de Nantes 12/4/INFO, 2012.

[18] Robert M. Haralick and Gordon L. Elliott. Increasing tree search efficiency for constraint satisfaction problems. In *Proceedings of the 6th international joint conference on Artificial intelligence - Volume 1*, IJCAI'79, pages 356–364. Morgan Kaufmann Publishers Inc., 1979.