Propagation de contraintes arithmétiques

Arnaud Malapert et Jean-Charles Régin
I3S CNRS – Université Nice-Sophia Antipolis
{arnaud.malapert,jean-charles.regin}@unice.fr

Résumé

Nous nous intéressons à la résolution de problèmes de grandes tailles contenant principalement des contraintes arithmétiques comme des problèmes de plus court chemin. Nous montrons qu'un simple modèle PPC n'est pas compétitif avec des algorithmes ou contraintes spécialisés. Ce phénomène est causé par le mécanisme de propagation de contraintes qui détermine l'ordre dans lequel les contraintes sont révisées. Fort de cette observation, nous proposons de modifier la propagation pour intégrer certaines idées cruciales, mais simples, des algorithmes spécialisés. Nous montrons l'intérêt de cette idée sur des problèmes de plus court chemin et nous questionnons sa généralisation à travers des expériences sur d'autres problèmes. Nous analysons aussi la dynamique du phénomène de propagation et constatons que le nombre de variables traitées plusieurs fois dans une même phase de propagation est faible.

Abstract

We consider the resolution by constraint programming of large problems, *i.e.* involving millions of constraints, which mainly imply arithmetic constraints, like shortest path problems or other related problems. We show that a simple constraint programming model is not competitive with dedicated algorithms (or dedicated constraints). This mainly comes from the propagation mechanism, *i.e.* the ordering along which the constraints are revised. Thus, we propose a modification of this propagation mechanism integrating the main ideas of the dedicated algorithms. We give some experiments for the shortest path problem and more general problems which confirms the robustness of our approach. Last, we give some results showing that only a few variables are considered more than once during a propagation step.

1 Introduction

Le problème du plus court chemin (PCC) est une composante de problèmes plus généraux comme les problèmes de cheminement avec contraintes de ressources ou des problèmes d'ordonnancement (PERT/CPM). Nous nous intéresserons à la variante qui consiste à trouver les chemins de longueurs (sommes des longueurs de leurs arcs) minimales d'un nœud racine s vers les autres nœuds dans un graphe orienté. Ce problème peut être résolu efficacement par une méthode de marquage (*labeling method*) utilisant une des deux stratégies classiques pour sélectionner la prochaine variable à examiner, celle de Bellman-Ford-Moore [2] et celle de Dijkstra [6]. Avant de résoudre des problèmes plus généraux, nous comparerons d'abord les performances de la programmation par contraintes (PPC) par rapport aux algorithmes spécialisés. Nous montrerons en fait que la conception d'un modèle simple compétitif avec ces algorithmes est loin d'être évidente. Ces piètres performances sont principalement dues au mécanisme de propagation de contraintes dont nous modifierons en conséquence l'initialisation et l'ordre de traitement des variables.. Puis, nous évaluerons l'impact de ces modifications sur les *radio link frequency assignment problems* (RLFAP). Finalement, nous analyserons des résultats concernant le nombre de fois où une variable est traitée pendant une phase de propagation.

2 Méthode de marquage et PPC

La méthode de marquage et la propagation de contraintes ont des ressemblances frappantes pour la résolution du PCC. Soit $G = (X, A)$ un graphe orienté où chaque arc $(u, v) \in A$ a une longueur $l(u, v)$, et soit s un nœud de X. Nous cherchons à calculer les plus courts chemins depuis ce nœud racine s vers tous les autres nœuds $v \in X$.

Méthode de marquage La méthode de marquage est définie de la manière suivante [5]. Pour chaque nœud v, la méthode maintient sa distance $d(v)$, son parent $p(v)$ et son statut $S(v) \in \{unreached, labeled, scanned\}$.

Lors de l'initialisation, ces valeurs sont fixées de la manière suivante pour chaque nœud $v \in X$: $d(v) = \infty$, $p(v) = null$ et $S(v) = unreached$. La première étape met à jour les informations sur le nœud racine : $d(s) = 0$ et $S(s) = labeled$. Ensuite, l'opération SCAN est appelée jusqu'à ce que tous les nœuds soient *scanned*. L'opération SCAN change éventuellement le statut *unreached* ou *scanned* des nœuds en *labeled*. En l'absence de cycle de coût négatif, la méthode de marquage s'achève et les parents définissent un arbre des plus courts chemins alors que $d(v)$ est la longueur du plus court chemin de s à v pour chaque nœud v.

Function SCAN(v)

for *each* $(v, w) \in A$ **do**
  if $d(v) + l(v, w) < d(w)$ **then**
    $d(w) \leftarrow d(v) + l(v, w)$;
    $S(w) \leftarrow labeled$;
    $p(w) \leftarrow v$;
$S(v) \leftarrow scanned$;

L'algorithme de Bellman-Ford maintient l'ensemble des noeuds marqués dans une queue de priorité FIFO : le prochain nœud à examiner est pris en tête alors qu'un nœud nouvellement marqué est ajouté en queue. L'algorithme de Dijkstra sélectionne le nœud marqué v dont la valeur $d(v)$ est minimale, mais ne peut traiter que des longueurs positives.

Propagation de contraintes D'un autre côté, la PPC associe un algorithme de filtrage FILTER(C) à chaque contrainte C, qui réduit les domaines des variables en supprimant les valeurs inconsistantes, *i.e.* qui ne peuvent pas appartenir à une solution de la contrainte. FILTER(C) renvoie l'ensemble des variables qui ont été modifiées durant l'étape de filtrage. Après chaque modification d'un domaine d'une variable, toutes les contraintes impliquant cette variable doivent être révisées à nouveau puisque de nouvelles réductions de domaine sont possibles. Ce processus est répété jusqu'à ce qu'aucune réduction ne soit plus possible ce qui arrive nécessairement puisque les domaines sont finis. La fonction FILTER(Q) met en œuvre le mécanisme appelé *propagation de contraintes*.

Function FILTER(Q)

while $Q \neq \emptyset$ **do**
  `/* pick` y `in` Q `and remove` y `from` Q `*/`
  for *each constraint* C *involving* y **do**
    $M \leftarrow$ FILTER(C) ;
    if $\exists x \in M / D(x) = \emptyset$ **then return** false;
    $Q \leftarrow Q \cup M$;
return true;

Les conclusions de plusieurs études [8, 3, 1] sur l'ordonnancement des révisions de variables/contraintes pour la consistance d'arc sont les suivantes. Une *propagation orientée par les variables* est préférable à une propagation orientée par les contraintes. La meilleure stratégie consiste à *sélectionner une variable dont la taille du domaine est minimale*. La propagation orientée variable sélectionne successivement les variables dont le domaine a été réduit et appelle les algorithmes de filtrages des contraintes impliquant cette variable. La propagation orientée par les contraintes considère les contraintes à tour de rôle sans se soucier des variables, comme l'algorithme AC-3. Les conclusions sont similaires pour les méthodes de marquage [7].

3 Modèle de plus court chemin

Nous décrivons maintenant un modèle PPC simple pour le problème de plus court chemin. Pour chaque nœud $v \in X$, soit $d(v)$ une variable entière positive ou nulle sauf pour le nœud racine où $d(s) = 0$. Pour chaque arc $(u, v) \in A$, nous posons une contrainte $d(v) \leq d(u) + l(u, v)$ qui réalise la *consistance de borne* plutôt que la consistance d'arc. Par conséquent, la borne supérieure du domaine de $d(v)$ après la propagation initiale est la plus courte distance d'un chemin partant de s (pas de recherche arborescente).
À première vue, nous pouvons simuler l'algorithme de Bellman-Ford en utilisant une queue FIFO Q et l'algorithme de Dijkstra en choisissant une variable de Q dont la taille du domaine est minimale (les bornes inférieures des domaines sont 0 et les bornes supérieures sont les distances depuis s). En réalité, cette affirmation est fausse pour presque tous les solveurs existants à cause de l'initialisation des contraintes, *i.e.* le premier appel à leurs algorithmes de filtrage. Nous allons voir que la plupart des solveurs ajoutent une nouvelle contrainte uniquement lorsque la propagation des contraintes du sous-problème courant est terminée.

4 Modification de la propagation

Initialisation des contraintes La plupart des solveurs gèrent uniquement les modifications entre deux appels consécutifs du même algorithme de filtrage. Cette information peut être exploitée par les algorithmes de filtrage, mais son maintien est compliqué par la propagation des autres contraintes/variables qui a eu lieu entre la modification du domaine et la propagation d'une variable. FILTER(C) utilise généralement les domaines courants quand la contrainte est initialisée, mais le filtrage de la contrainte C peut alors être appelé plusieurs fois pour les mêmes modifications si plusieurs variables de C sont dans Q. Dans ce cas, le solveur applique FILTER(C) pour des modifications

antérieures à l'initialisation de la contrainte, ce qui est déraisonnable. Une solution simple adoptée dans la plupart des solveurs consiste donc à retarder l'initialisation d'une contrainte tant que le sous-problème courant n'a pas été entièrement propagé.
Cette solution a de sérieuses conséquences illustrées sur le PCC décrit en figure 1 où les arcs sont étiquetés par leur longueur et leur contrainte. Un solveur ajoutera les contraintes à tour de rôle et propagera les modifications sur le sous-problème courant. Les contraintes seront ajoutées ou filtrées dans l'ordre suivant : $C_1, C_2, C_3, C_4, C_5, C_6$ (v_1 est réduite), C_2 (v_2 est réduite), C_4, C_7, C_8, C_9 (v_1 est réduite), C_2 (v_2 est réduite), C_4, C_{10}, C_{11}, C_{12} (v_1 est réduite), C_2, et C_4. La méthode de marquage insérera s dans la queue. Les contraintes $C_1, C_3, C_5, C_8, C_{11}$ seront traitées depuis s et ajouterons les nœuds v_1, v_2, v_3, v_4, v_5 dans la queue. Ensuite, v_1 est extraite ; C_2 est filtrée ; v_2 est extraite ; C_4 est filtrée ; v_3 est extraite ; C_6 est filtrée (ajoute v_1) ; C_7, C_9, C_{10}, C_{12} sont filtrées ; v_1 est extraite ; C_2 est filtrée (ajoute v_2) ; v_2 est extraite et C_4 est filtrée. La méthode de marquage examine moins de contraintes/arcs plusieurs fois : les contraintes C_2 et C_4 ne sont filtrées que deux fois au lieu de quatre.

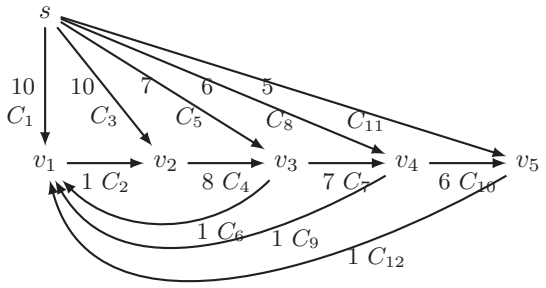

FIGURE 1 – Un exemple de problème de plus court chemin.

Pour résoudre ce problème, nous avons séparé l'initialisation de la propagation ce qui permet d'ajouter de nouvelles contraintes sans attendre la fin de la propagation du sous-problème courant. Nous dirons que la propagation est immédiate quand la fonction FILTER(Q) est appelée dès qu'une modification a eu lieu.

Heuristique des parents L'algorithme de Bellman-Ford peut être amélioré en utilisant l'heuristique des parents (*parent checking*) qui examine un nœud v si son parent $p(v)$ n'appartient pas à Q. Dans un problème de plus court chemin, la présence du parent d'un nœud v dans la queue Q entraînera nécessairement une nouvelle modification de $d(v)$. Même si cela n'est pas nécessairement le cas pour la propagation de contraintes, nous évaluerons une nouvelle variante où la propagation d'une variable est retardée tant que son parent appartient à la queue Q.

5 Expérimentations

Nous récapitulons les résultats obtenus pour plusieurs variantes de la propagation de contraintes implantées dans le solveur Choco (`http://choco.mines-nantes.fr`). Soit P** et _** les variantes avec et sans vérification des parents. Soit *F* et *D* les variantes où la sélection de la prochaine variable suit une politique FIFO ou du domaine minimal. Soit **_ et **S les variantes avec propagation immédiate ou initialisation séparée. La sélection d'une variable a une complexité au plus linéaire en fonction de $|Q|$.

Plus court chemin Les résultats sur les problèmes de plus court chemin donnés dans le tableau (a) concernent quatre catégories de graphes générés aléatoirement contenant 2000 nœuds et approximativement 2 millions de contraintes. Les catégories D et SD définissent des graphes orientés acyclique complet ou semi-complet avec des longueurs positives et 20% d'arcs supplémentaires. Dans un graphe semi-complet, les degrés entrants et sortants de la moitié des nœuds sont unitaires. Chaque arc supplémentaire crée un cycle de longueur positive. Les catégories DN et SDN se distinguent de D et SD par la présence d'arcs de longueur négative, mais ne créant aucun cycle de longueur négative. Pour chaque catégorie, les lignes $\#v$ et t correspondent au nombre de variables propagées et à la durée de propagation en secondes.
La politique du domaine minimal est plus efficace que la politique FIFO et l'initialisation séparée se révèle intéressante. La combinaison des deux simule parfaitement l'algorithme de Dijkstra même en présence de longueurs négatives. Par conséquent, la propagation des variables est « minimale » puisque chaque variable est propagée une seule et unique fois (toutes les variables sont modifiées). Par contre, la politique du domaine minimal est plus lente que FIFO probablement à cause de la taille de Q sauf pour la catégorie DN.
L'heuristique des parents offre une légère amélioration par rapport à *F_ en retardant approximativement la propagation de 1% des variables. Par contre, elle ralentit significativement la propagation de *FS parce que 135% des variables de Q sont retardées (certaines variables sont retardées plusieurs fois) et Q est de grande taille. Nous avons aussi évalué des variantes de l'heuristique des parents (un ancêtre est dans la queue, plusieurs parents ...) sans noter d'amélioration particulière. En conclusion, ces résultats montrent que l'heuristique des parents a un intérêt limité pour la PPC.

Affectation de fréquences radio Les modèles du RLFAP sont constitués de variables (6000 en moyenne) avec des domaines énumérés et de contraintes arith-

		_F_	_FS	_D_	_DS	PF_	PFS	PD_	PDS
D	#v	10516	7551	9164	1999	10343	7480	9164	1999
	t	16.4	3.2	14.7	4.8	16.3	12.2	3.4	5.6
SD	#v	10231	7888	9164	1999	10091	7772	9164	1999
	t	6.9	1.8	6.4	2.7	6.8	5.5	1.9	3.3
DN	#v	13916	7177	12506	1999	13732	7146	12506	1999
	t	19.1	12.0	17.1	6.0	18.8	13.5	16.1	6.7
SDN	#v	13021	7328	11885	1999	12867	7291	11885	1999
	t	7.6	1.7	7.1	2.7	7.6	5.2	2.1	3.4

(a) Résultats pour les problèmes de plus court chemin.

	_F_	_FS	_D_	_DS
%v	101	92	99	84
%r	5	2	4	1
t	0.67	0.78	0.70	3.30

	_F_		_D_	
	‰v	‰r	‰v	‰r
SAT	30.1	1.5	28.6	0.1
UNSAT	4.6	0.5	1.9	0.1

(b) RLFAP : Propagation initiale et shaving

métiques imposant la consistance d'arc ou de borne. Le tableau supérieur (b) récapitule les résultats de la propagation initiale. Les variantes avec l'heuristique des parents n'y apparaissent pas car elles n'apportent aucune amélioration même si 13% et 89% des variables propagées sont retardées pour PF_ et PFS. Les lignes %v, %r et t représentent respectivement le pourcentage de variables propagées par rapport au nombre total de variables, le pourcentage de variables ré-entrantes par rapport au nombre de variables propagées et la durée de propagation en secondes. **S réduit le nombre de variables propagées et ré-entrantes sans toutefois améliorer la durée de propagation surtout pour _DS. La propagation peut être améliorée en réduisant le nombre de fois où une même variable est propagée. Chaque variable modifiée doit être extraite au moins une fois de la queue Q puisqu'il est nécessaire d'étudier les conséquences de sa modification. Donc, une « propagation parfaite » ne propagerait au mieux qu'une seule fois chaque variable modifiée. Ainsi, *le nombre de variable ré-entrantes est une borne supérieure sur la diminution du nombre de variables traitées pendant une phase de propagation par une autre politique d'ordonnancement.*

Le tableau inférieur (b) donne les résultats d'une phase de *shaving* sur les bornes des domaines. **S n'y apparaît pas puisque les contraintes ont déjà été initialisées. Les lignes ‰v et ‰r correspondent respectivement au pour-mille des variables propagées et ré-entrantes. Les temps de calcul ne sont pas mentionnés, car le solveur a été lourdement instrumenté. Les résultats sont regroupés selon la réponse donnée par la propagation (550000 SAT, 130000 UNSAT). Les réponses SAT où une seule variable est propagée ont été ignorées. Le nombre de variables propagées est légèrement réduit grâce à _D_, surtout pour les réponses UNSAT. *Le nombre de variables ré-entrantes est très faible ce qui limite l'amélioration espérée en utilisant un autre ordonnancement des variables.* Ces résultats confirment les observations de [4] : *moins de variables sont propagées pour une réponse UNSAT que pour SAT.*

6 Conclusion

Nous avons montré que les solveurs doivent être modifiés pour être compétitifs avec les meilleurs algorithmes pour résoudre les problèmes de plus court chemin. Nous avons modifié le mécanisme de propagation de contraintes pour le rendre plus efficace sur des problèmes tels que PCC ou RLFAP. Malheureusement, nous avons aussi montré qu'un nombre restreint de variables sont traitées plusieurs fois pendant une phase de propagation ce qui limite l'amélioration induite par un autre ordonnancement des variables.

Références

[1] T. Balafoutis and K. Stergiou. Exploiting constraint weights for revision ordering in AC algorithms. In *Proc. of ECAI-2008-W31*, 2008.

[2] R. Bellman. On a Routing Problem. *Quarterly of Applied Mathematics*, 16 :87–90, 1958.

[3] F. Boussemart, F. Hemery, and C. Lecoutre. Revision ordering heuristics for the CSP. In *Proc. of CPAI 2004*, pages 29–43, 2004.

[4] F. Boussemart, F. Hemery, C. Lecoutre, and M. Samy-modeliar. Contrôle statistique du processus de propagation de contraintes. In *Proc. of JFPC 2011*, pages 65–74, 2011.

[5] B.V. Cherkassky, A.V. Goldberg, and T. Radzik. Shortest paths algorithms : Theory and experimental evaluation. *Math. Program.*, 73 :129–174, 1996.

[6] E.W. Dijkstra. A note on two problems in connection with graphs. *Num. Math.*, 1 :269–271, 1959.

[7] Y. Dinitz and R. Itzhak. Hybrid bellman-ford-dijkstra algorithm. Technical Report CS-10-04, Ben-Gurion University of the Negev, 2010.

[8] R.J. Wallace and E.C. Freuder. Ordering heuristics for arc consistency algorithms. In *Proc. of Canadian AI 1992*, pages 163–169, 1992.