Réparation de solutions par propagation de réseaux d'inégalités dans LocalSolver

Léa Blaise[1,2]

[1] LAAS-CNRS, Université de Toulouse, CNRS, INP, Toulouse, France
[2] LocalSolver, 36 Avenue Hoche, Paris, France
lblaise@localsolver.com

Mots-clés : *propagation de contraintes, réseaux d'inégalités, recherche locale, réparation de solutions, solveur, ordonnancement*

1 Introduction et contexte

LocalSolver est un solveur d'optimisation mathématique basé sur différentes techniques de recherche opérationnelle, dont la recherche locale [2], mais aussi la programmation linéaire, non linéaire, et par contraintes. Son but est d'offrir une approche de type "model-and-run" à des problèmes d'optimisation (combinatoires, continus, mixtes...), y compris sur de grandes instances.

On s'intéresse ici à des problèmes structurés par des réseaux d'inégalités, dont les contraintes comprennent des inégalités linéaires à deux variables, ou des disjonctions de telles inégalités. Parmi ces problèmes, on peut trouver de nombreux problèmes d'ordonnancement, comme le Job Shop [1], mais aussi des problèmes de packing, de layout, ou de mining.

Seuls, les algorithmes de recherche locale à petits voisinages se retrouvent en difficulté face à ces problèmes. En effet, ces contraintes (contraintes de précédence et de ressource disjonctive dans le cas du Job Shop) sont très serrées dans une bonne solution. Le passage d'une bonne solution à une autre nécessite donc de faire de petits changements sur de nombreuses variables entières (dates de début des tâches), ce qui est difficilement envisageable avec des techniques de recherche locale à petits voisinages.

2 Réparation de solutions

La solution que nous avons envisagée et implémentée dans LocalSolver pour pallier ce problème consiste en une propagation des contraintes, qui permet de réparer de proche en proche, une contrainte à la fois, une solution prometteuse mais non réalisable.

Avant de commencer à résoudre, certaines contraintes spécifiques (notamment contraintes de précédence généralisées et de ressource disjonctive) sont détectées dans le modèle : ce sont celles qui seront réparées. On considère une itération de la recherche locale : on dispose d'une solution réalisable, sur laquelle on applique un mouvement, la rendant ainsi non réalisable ($\mathcal{S}$). On applique alors la phase de réparation des contraintes, semblable dans le cas le plus générique à de la propagation en Programmation par Contraintes [3], avec toutefois quelques différences notables. Tout d'abord, on ne propage pas les réductions des domaines, mais uniquement des modifications des supports des variables : le but étant de trouver une solution réalisable proche de $\mathcal{S}$, on ne propage que les contraintes violées, en modifiant les valeurs des variables pour les réparer. De plus, on impose de toujours modifier les valeurs des variables dans le même sens : on pourra augmenter davantage la valeur d'une variable que l'on a déjà augmentée à cette itération, mais on ne pourra plus la diminuer, et réciproquement. Les réparations suivent donc la "direction" prise par le mouvement : on répare la solution en amplifiant le mouvement plutôt qu'en l'annulant.

2.1 Inégalités linéaires à deux variables

On considère des inégalités de la forme

$$aX + bY \leq c$$

où X et Y sont des variables entières ou flottantes, et a, b, et c des constantes quelconques. Ces inégalités peuvent notamment représenter des contraintes de précédence généralisées (avec $a = 1$ et $b = -1$), mais la réparation n'est pas limitée à ce cas particulier.

On suppose que l'inégalité $aX+bY \leq c$ est violée. Puisque la solution initiale était réalisable, au moins l'une des variables a déjà bougé dans la "mauvaise" direction, et ne peut donc plus bouger dans la direction de la réparation. Il n'y a donc qu'une possibilité pour réparer la contrainte : modifier l'autre variable, juste assez pour réparer la contrainte ($X \leftarrow \frac{c-bY}{a}$).

Lorsque les contraintes réparables sont des inégalités, la phase de réparation équivaut à une propagation particulière des contraintes, dans laquelle on ne propage la réduction d'une borne d'une variable que si elle exclut son support. Puisque chaque variable doit toujours bouger dans le même sens, l'une de ses bornes seulement peut être modifiée au cours de la propagation.

2.2 Disjonctions d'inégalités linéaires à deux variables

On considère des disjonctions d'inégalités de la forme

$$\bigvee_i (a_i X_i + b_i Y_i \leq c_i)$$

où les X_i et Y_i sont des variables entières ou flottantes, et les a_i, b_i, et c_i des constantes quelconques. Ces inégalités peuvent notamment représenter des contraintes de ressource disjonctive ou de packing (avec $a_i = 1$ et $b_i = -1$ $\forall i$), mais la réparation n'est pas limitée à ce cas particulier.

La réparation d'une disjonction violée est non déterministe. D'abord, on choisit au hasard quelle inégalité de la disjonction sera réparée (si elle n'est pas réparable, on passe à la suivante, et ainsi de suite). Ensuite, si les deux variables de l'inégalité choisie peuvent bouger dans le sens de la réparation (ce qui est possible dans le cas d'une disjonction), on choisit au hasard la façon dont on la répare. Quatre possibilités existent pour cela : on répare en bougeant seulement X ou seulement Y, ou en attribuant des parts équitables ou aléatoires de la réparation à X et Y.

3 Résultats

L'intégration de ce mécanisme de réparation de solutions par propagation améliore considérablement les performances de LocalSolver sur les problèmes visés.

On peut notamment le constater sur les instances de la classe FT (Fisher et Thompson [1]) du problème du Job Shop. Les résultats obtenus sans ce mécanisme sont en moyenne à 73% de l'optimum au bout de 10 secondes de recherche, et à 15% au bout de 60 secondes, contre 7% en 10 secondes et 4% en 60 secondes en ajoutant le mécanisme de réparation.

La réparation des solutions permet aussi d'obtenir de meilleurs résultats sur des problèmes de packing 3D et de mining, notamment sur des instances industrielles de notre base de tests.

Références

[1] H. Fisher and G. Thompson. *Probabilistic learning combinations of local job-shop scheduling rules*. Industrial Scheduling, Muth J., G. Thompson (Eds.), Prentice Hall, Englewood Cliffs, New Jersey, pp. 225-251, 1963.

[2] F. Gardi, T. Benoist, J. Darlay, B. Estellon, and R. Megel. *Mathematical Programming Solver Based on Local Search*, Wiley, 2014.

[3] F. Rossi, P. Van Beek, and T. Walsh. *Handbook of Constraint Programming (Foundations of Artificial Intelligence)*, Elsevier Science Inc., New York, NY, USA, 2006.