Panoramyx : une bibliothèque pour le développement de solveurs de contraintes parallèles

Thibault Falque, Jean-Marie Lagniez, Romain Wallon

Panoramyx : une bibliothèque pour le développement de solveurs de contraintes parallèles

Thibault Falque[1,2], Jean-Marie Lagniez[2], Romain Wallon[2]

[1] Exakis Nelite, France

[2] CRIL, Université d'Artois & CNRS, 62300 Lens, France

{falque,lagniez,wallon}@cril.univ-artois.fr

Mots-clés : *programmation par contraintes, problème de cohérence propositionnelle, raisonnement pseudo-booléen, solveurs parallèles, solveurs distribués.*

De nombreux problèmes, industriels ou académiques, peuvent être représentés sous la forme de conjonctions de contraintes. La programmation par contraintes [9, 4] consiste à modéliser ces problèmes (dans notre cas, en utilisant le langage de modélisation `XCSP3`), pour ensuite pouvoir les résoudre à l'aide de solveurs raisonnant sur la structure du problème considéré, indépendamment de sa sémantique. Nous pouvons notamment citer `CoSoCo` ou `ACE` comme exemples de tels solveurs. Ceux-ci sont souvent capables de répondre à différents types de requêtes. Le problème de satisfaction de contraintes (CSP) consiste à déterminer si un ensemble de contraintes est cohérent et, si tel est le cas, d'identifier une solution satisfaisant l'ensemble de ses contraintes, voire d'énumérer l'ensemble des solutions. Le problème d'optimisation sous contraintes (COP) consiste quant à lui à trouver une solution à un ensemble de contraintes qui minimise (ou maximise) la valeur d'une (ou plusieurs) fonction(s) objectif donnée(s).

Bien que ces solveurs fournissent souvent des performances satisfaisantes, certains problèmes restent difficiles à résoudre dans un temps raisonnable. Ceci est d'autant plus vrai pour les problèmes d'énumération ou d'optimisation, qui demandent d'explorer l'intégralité de l'espace de recherche. Pour résoudre ce type de problèmes, il peut être tentant d'exploiter le fonctionnement des processeurs modernes et le *cloud computing* pour développer des approches de résolution parallèles ou distribuées.

Parmi les approches ayant été proposées, nous pouvons citer les approches de type *portfolio*, dans lesquelles plusieurs solveurs différents sont exécutés en parallèle sur la même entrée, afin d'exploiter leur complémentarité. Une autre apprroche, appelée *Embarrassingly Parallel Search* (EPS), consiste à affecter différentes parties de l'espace de recherche à différents solveurs (ou à plusieurs instances du même solveur) en leur attribuant des affectations partielles complémentaires [5]. Les solveurs utilisés dans ces approches peuvent travailler indépendamment, ou communiquer. Par exemple, dans le cas de problèmes d'optimisation, les bornes obtenues peuvent être partagées entre les différents solveurs. Une autre possibilité est d'utiliser le vol de travail (*work stealing*), dans lequel un solveur « en difficulté » peut séparer son espace de recherche en plusieurs sous-espaces de recherche, qui seront alors récupérés par des solveurs disponibles, comme cela a par exemple été proposé dans [1]. Ces approches sont implantées dans des solveurs comme `Gecode`, `Toulbar2`, `Choco` [8] ou `OR-Tools` [7].

Cependant, ces approches peuvent être difficiles à mettre en œuvre, car elles impliquent des mécanismes de synchronisation et de gestion de ressources complexes. Afin de simplifier le développement de tels solveurs, différents *frameworks* ont été proposés. Nous pouvons par exemple citer `PaInleSS` [3] pour le paradigme SAT, ou encore `Bob++` [2, 6] dans le cas de la programmtion par contraintes, construit sur les solveurs `Gecode` et `OR-Tools`. Ces bibliothèques sont conçues pour fournir les briques élémentaires nécessaires au développement de solveurs parallèles ou distribués, pour permettre de se concentrer sur le développement de nouvelles stratégies.

Dans la même optique, nous présentons PANORAMYX (*Programming pArallel coNstraint sOlveRs mAde aMazingly easY*), une bibliothèque C++ conçue pour être multi-plateforme et supporter une large variété de solveurs, pouvant être implantés dans différents langages de programmation (pour l'instant C, C++ et Java grâce à JNI).

Plus précisément l'intégration de différents solveurs de l'état de l'art est rendue possible grâce à l'interface UNIvERSE (*mUlti laNguage unIfied intErface foR conStraint solvErs*). Celle-ci supporte différents paradigmes de résolution, en fournissant des interfaces permettant d'adapter n'importe quels solveurs SAT, pseudo-booléens ou CP, en proposant à la fois des méthodes pour ajouter des contraintes aux solveurs et pour appliquer des approches de résolution incrémentale. Ces solveurs peuvent ensuite être utilisés indistinctement comme moteur de résolution dans PANORAMYX.

PANORAMYX fournit également un certain nombre d'interfaces définissant différentes stratégies de répartition des tâches, de partage d'informations et de synchronisation entre les solveurs, entre autres. Celles-ci peuvent être implantées indépendamment, afin de pouvoir facilement tester de nouvelles approches parallèles, ou encore d'implanter des approches existantes comme le *portfolio* ou d'*EPS*, en permettant dans ce dernier cas d'utiliser différentes stratégies de décomposition de l'espace de recherche.

L'échange d'informations entre les solveurs repose quant à lui sur la bibliothèque MPI (*Message Passing Interface*), qui permet de gérer à la fois des approches parallèles ou distribuées, de manière transparente. Elle peut toutefois être facilement remplacée par d'autres mécanismes (comme les threads POSIX par exemple), là encore grâce à l'utilisation d'interfaces. Le choix des informations à partager est laissé à une stratégie indépendante, afin de pouvoir facilement prototyper de nouvelles idées.

Dans cette présentation, nous introduirons les différentes interfaces fournies par PANORAMYX, en illustrant leur implantation avec des approches de type portfolio ou EPS, en utilisant différents solveurs et différentes stratégies.

Références

[1] Geoffrey Chu, Christian Schulte, and Peter J. Stuckey. Confidence-based work stealing in parallel constraint programming. In *Proceedings of CP 2009*, pages 226–241, 2009.

[2] François Galea and Bertrand LeCun. Bobpp : a framework for exact combinatorial optimization methods on parallel machines. *PGCO'2007*, pages 779–, 05 2007.

[3] Ludovic Le Frioux, Souheib Baarir, Julien Sopena, and Fabrice Kordon. PaInleSS : A framework for parallel SAT solving. In *Proceedings of SAT 2017*, pages 233–250, 2017.

[4] C. Lecoutre. *Constraint Networks : Techniques and Algorithms*. ISTE/Wiley, 2009.

[5] Arnaud Malapert, Jean-Charles Régin, and Mohamed Rezgui. Embarrassingly Parallel Search in Constraint Programming. *JAIR*, 57 :421 – 464, 2016.

[6] Tarek Menouer and Bertrand LeCun. A parallelization mixing or-tools/gecode solvers on top of the bobpp framework. pages 242–246, 10 2013.

[7] Laurent Perron and Vincent Furnon. Or-tools.

[8] Charles Prud'homme, Jean-Guillaume Fages, and Xavier Lorca. Choco solver documentation. *TASC, INRIA Rennes, LINA CNRS UMR*, 6241, 2016.

[9] F. Rossi, P. V. Beek, and T. Walsh. *Handbook of Constraint Programming*. Elsevier, 2006.