Partitionnement d'Hypergraphes pour la Compilation de Formules Pseudo-Booléennes

Romain Wallon

LIX (Laboratoire d'Informatique de l'X), École Polytechnique, Chaire X-Uber
wallon@lix.polytechnique.fr

Mots-clés : *contraintes pseudo-booléennes, compilation de connaissances, partitionnement d'hypergraphes.*

De nombreux problèmes en intelligence artificielle peuvent être exprimés comme des conjonctions de contraintes en variables booléennes, pouvant prendre la valeur 0 (faux) ou 1 (vrai). Pour représenter ce type de problèmes, il est fréquent d'utiliser des formules en forme normale conjonctive (CNF). En effet, grâce aux solveurs SAT dits « modernes » [11, 12, 5], il est en pratique possible de raisonner efficacement sur une large variété d'instances encodées sous cette forme. Ces solveurs sont cependant basés sur le système de preuve par résolution, et sont donc limités par les faiblesses de ce système. En particulier, les solveurs SAT modernes ne parviennent que difficilement à raisonner sur des formules nécessitant de « savoir compter » (comme par exemple les formules représentant le principe dit *du pigeonnier* [7]). C'est ce qui a motivé le développement de solveurs pseudo-booléens (PB) [4, 13, 10, 6], qui permettent non seulement de gérer les contraintes plus expressives que sont les contraintes de cardinalité et les contraintes PB (qui autorisent l'utilisation de coefficients), mais également d'utiliser le système de preuve des plans-coupes, plus puissant que la résolution.

Néanmoins, l'efficacité pratique des solveurs SAT et le pouvoir d'inférence des solveurs PB ne suffisent pas à garantir des temps de réponse raisonnables pour les problèmes NP-complets qu'ils doivent résoudre. Dans certaines applications, notamment lorsque des interactions avec l'utilisateur sont attendues, cela ne peut être accepté. Une solution est alors de passer par une phase de *compilation* [1], consistant à traduire les formules considérées dans un langage différent, dans lequel les opérations envisagées peuvent être réalisées efficacement (idéalement, en temps polynomial). Dans ce contexte, il est commode de représenter les formules considérées sous la forme de graphes ou d'hypergraphes, lesquels fournissent des informations importantes quant à la structure de la formule. Les compilateurs actuels exploitent ainsi le partitionnement de ces graphes pour choisir comment affecter certaines variables afin d'obtenir des sous-formules indépendantes [2, 8].

Dans cette présentation, nous étudierons dans un premier temps les caractéristiques des formules CNF, ainsi que les différents graphes utilisés pour les représenter. En particulier, nous évoquerons le cas du *graphe primal* de ces formules (dont les nœuds correspondent aux variables de la formule, et les arêtes relient des variables apparaissant conjointement dans une même contrainte) ainsi que celui de leur *graphe dual* (dont les nœuds sont les variables ou les contraintes, et les variables sont reliées par une arête aux contraintes dans lesquelles elles apparaissent). Nous introduirons également les hypergraphes équivalents utilisés dans le cadre de la compilation de formules booléennes.

Dans un second temps, nous présenterons les enjeux de cette compilation, notamment à la lumière des critères de la *carte de compilation* [3]. Nous considérerons en particulier la compilation de formules CNF dans le langage des Decision-DNNF, en présentant l'algorithme du compilateur *D4* [8]. Nous insisterons notamment sur la manière dont un partitionnement de l'hypergraphe dual peut être utilisé *pendant* cette compilation, et comment la qualité du partitionnement obtenu joue un rôle dans la taille de la Decision-DNNF produite.

Enfin, nous étudierons dans un troisième temps comment l'algorithme de compilation de formules CNF sus-mentionné peut être étendu à la compilation native de formules PB. Cette dernière n'a pour l'instant été que peu étudiée, en particulier parce que les solveurs PB sont généralement moins efficaces en pratique que les solveurs SAT classiques (en raison des structures de données supplémentaires qu'ils doivent manipuler). Toutefois, des avancées récentes dans l'implantation de solveurs PB [6, 9] ont contribué à rendre ces derniers suffisamment efficaces pour pouvoir les utiliser dans une phase de compilation. Cette possibilité pourrait alors permettre de compiler des formules pour lesquelles un encodage CNF rendrait leur compilation inenvisageable (à la fois en termes de temps et d'espace mémoire nécessaires).

Remerciements Cette présentation bénéficie du soutien de la Chaire « Integrated Urban Mobility » portée par l'X - Ecole Polytechnique et La Fondation de l'Ecole Polytechnique, et soutenue par Uber. La responsabilité des Partenaires de la Chaire ne peut en aucun cas être mise en cause en raison du contenu de la présente communication, qui n'engage que son auteur.

Références

[1] Marco Cadoli and Francesco M. Donini. A Survey on Knowledge Compilation. *AI Commun.*, 10(3,4) :137–150, December 1997.

[2] Adnan Darwiche. New advances in compiling cnf to decomposable negation normal form. In *Proceedings of the 16th European Conference on Artificial Intelligence*, ECAI'04, page 318–322, NLD, 2004. IOS Press.

[3] Adnan Darwiche and Pierre Marquis. A Knowledge Compilation Map. *J. Artif. Int. Res.*, 17(1) :229–264, September 2002.

[4] Heidi E. Dixon and Matthew L. Ginsberg. Inference methods for a pseudo-boolean satisfiability solver. In *AAAI'02*, pages 635–640, 2002.

[5] Niklas Eén and Niklas Sörensson. An extensible sat-solver. In *Theory and Applications of Satisfiability Testing*, pages 502–518, 2004.

[6] Jan Elffers and Jakob Nordström. Divide and conquer : Towards faster pseudo-boolean solving. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*, pages 1291–1299, 2018.

[7] Armin Haken. The intractability of resolution. *Theoretical Computer Science*, 39 :297 – 308, 1985. Third Conference on Foundations of Software Technology and Theoretical Computer Science.

[8] Jean-Marie Lagniez and Pierre Marquis. An improved decision-dnnf compiler. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*, pages 667–673, 2017.

[9] Daniel Le Berre, Pierre Marquis, and Romain Wallon. On weakening strategies for PB solvers. In Luca Pulina and Martina Seidl, editors, *Theory and Applications of Satisfiability Testing - SAT 2020 - 23rd International Conference, Alghero, Italy, July 3-10, 2020, Proceedings*, volume 12178 of *Lecture Notes in Computer Science*, pages 322–331. Springer, 2020.

[10] Daniel Le Berre and Anne Parrain. The SAT4J library, Release 2.2, System Description. *Journal on Satisfiability, Boolean Modeling and Computation*, 7 :59–64, 2010.

[11] Joao Marques-Silva and Karem A. Sakallah. Grasp : A search algorithm for propositional satisfiability. *IEEE Trans. Computers*, pages 220–227, 1999.

[12] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff : Engineering an Efficient SAT Solver. In *Proceedings of the 38th Annual Design Automation Conference*, DAC '01, pages 530–535, New York, NY, USA, 2001. ACM.

[13] Hossein M. Sheini and Karem A. Sakallah. Pueblo : A Hybrid Pseudo-Boolean SAT Solver. *JSAT*, pages 165–189, 2006.