A wonderful SAT solver
C++ mieux à deux

Rédouane Elghazi et Loïs Paulin

À Olga

Abstract. Nous prsentons ici un solveur SAT développé au cours de l'UE Projet2 dirigée par Daniel Hirschkoff. Le but était d'implémenter l'algorithme DPLL et d'y rajouter des modules comprenant des heuristiques et méthodes d'optimisation des déductions. Nous présenterons ici nos choix et les problèmes auxquels nous avons été confrontés.

1. Langage choisi

Nous avons choisi pour ce projet de coder en C++, car même si l'on avait déjà une bonne expérience du C pour l'un et en Python pour l'autre, nous avons pensé qu'il serait bon d'aquérir de l'experience dans un langage objet aussi utilisé que le C++, bon compromis entre les performances C et la programmation objet du Python.
De plus le C++ apporte de nombreux avantages dans ce projet : surcharge des opérateurs, librairie standard, performance d'un langage compilé (par rapport au Python).

2. Structures de Données

2.1. Représentation arborescente des formules

On a ici une classe représentant les formules sous forme d'arbre où les noeuds représentent les opérateurs et les sous-arbres les opérandes.
Cette structure est utilisée comme format de sortie du parser et entrée de la transformée de tseitin. Cependant, elle ne permet pas un parcours et une évolution performants de la formule.

2.2. Représentation ensembliste des formules

Afin de palier à ce manque de performance, la transformation de tseitin nous renvoit une représentation de la formule sous forme ensembliste, un tableau d'ensembles représentants chaque clauses. Ce choix ce justifie par un besoin de performance dans la recherche d'un élément dans une clause. les clauses actives sont représentée par un ensemble de façon à pouvoir supprimer des éléments en O(log n).

2.3. Myv

Ceci est une structure permettant d'utiliser des vecteurs avec des indices négatifs afin d'éviter l'utilisation de map lorsque l'on en utilise pas les avantages.

2.4. Union Find

L'union find avec compression nécessite la mise un place d'un arbre parcouru d'une feuille vers la racine. On représentera cet arbre par un tableau contenant le parent de chaque variable. Les classes d'équivalence sont représentées par un arbre, compressé à chaque parcours de façon à ce que la racine soit le plus proche possible des feuilles, ce qui permet en amorti de vérifier l'équivalence en O(1).

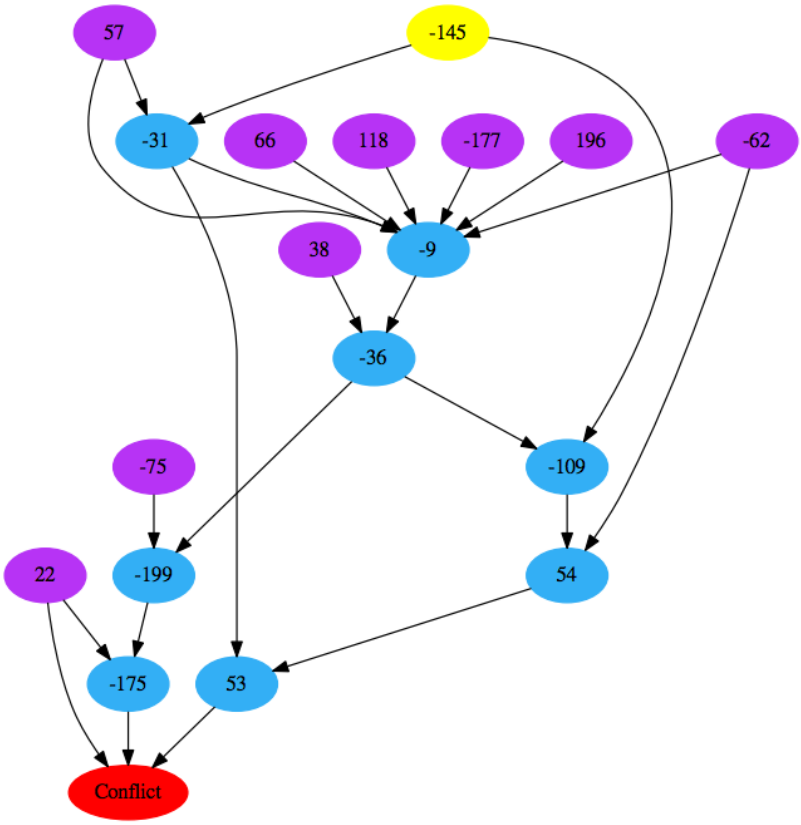
3. Backtrack

Le Backtrack a été mis en place par une classe contenant une pile d'éléments permettant de retenir et restaurer toutes les informations. Le backtrack retient dans une pile les variables affectées, et les clauses modifiées et supprimées par cette affectation. Afin d'éviter un cot de copie non nécessaire, le backtrack repose sur un système astucieux de pointeurs vers des tableaux représentant les clauses dans lesquels les variables apparaissent. On retient également si une variable à été choisie ou bien forcée de façon à backtrack jusqu'au dernier choix effectué.

4. Mode interactif

L'implémentation du mode intéractif à posé des problèmes car la gestion des entrées et de lecture du fichier n'était pas très propre. Après une correction des bugs le menu interractif fonctionne, et l'on peut sauvegarder les graphes de conflits [Figure 1] avant de les fermer.
Dans le graphe les littéraux ajouté dans la clause apprise apparaissent en violet, les littéraux du niveau courant en bleu, et l'UIP en jaune.

FIGURE 1. Exemple de graphe de conflit



5. Multithreading

Après un départ non fructueux de parallèlisme sur les watched litterals, nous avons décidé de faire un multithreading faisant tourner 4 heuristiques en parallèle et prend les résultats de la plus rapide, cela nous permet, en choisissant correctement les heuristique, ceci permet un gain sur le temps d'excution moyen d'environ 60% sur nos jeu de test 200 variables 860 clauses [2].

6. Performances

Les performances de notre solveur sont derrière minisat sur une partie des tests, cependant il y a plusieurs tests où l'on arrive à descendre en dessous de MiniSat [1]. Tous nos tests sont effectués sur les jeux de test 3-SAT de SATLIB [2].

References

[1] Niklas Eén and Niklas Sörensson. “Theory and Applications of Satisfiability Testing: 6th International Conference, SAT 2003, Santa Margherita Ligure, Italy, May 5-8, 2003, Selected Revised Papers”. In: (2004). Ed. by Enrico Giunchiglia and Armando Tacchella, pp. 502–518.

[2] Holger H Hoos and Thomas Stützle. “Satlib–the satisfiability library”. In: *Web site at: http://www.satlib.org* (1999).

Rédouane Elghazi

Loïs Paulin