

OpenSAT : Une plate-forme SAT Open Source

Gilles Audemard Daniel Le Berre
Olivier Roussel*

CRIL, Université d'Artois, Lens France

{audemard, leberre, roussel}@cril.univ-artois.fr

Résumé

Cet article présente OpenSAT, une plate-forme de développement *open source* pour le développement d'algorithmes reliés à la satisfaisabilité de formules propositionnelles. L'objectif de cette plate-forme est de favoriser le développement de nouveaux algorithmes en réutilisant les structures de données et les algorithmes qui ont prouvé leur efficacité dans différents prouveurs. Cette réutilisation se fonde sur l'utilisation d'un langage à objet, largement disponible et portable : Java. Ce type de plate-forme permet de fédérer les efforts de développement et de maintenance et de bénéficier de techniques avancées pour le prototypage de prouveurs. Chacun peut bénéficier des outils de cette plate-forme et y apporter ses propres améliorations en participant à son développement selon le modèle *open source*.

1 Introduction

Les dernières années ont constamment vu évoluer les prouveurs SAT [2, 10, 13, 16, 15, 19]. Ceux-ci sont maintenant capables de résoudre de grandes instances et des problèmes difficiles du "monde réel" (instances de model-checking [5], de planification [11]...). La recherche appliquée sur les prouveurs SAT est un domaine très attractif, de nouvelles idées (d'implantation, d'algorithmes, de structures de données) sont constamment proposées. Un témoignage de cette évolution est la compétition SAT2003¹ qui met en scène quelques 30 prouveurs cette année, alors qu'une autre compétition avait déjà été organisée l'année dernière et avait déjà réuni une trentaine d'autres prouveurs [17].

Ce scénario pose quelques problèmes lorsque l'on veut évaluer les performances et les mérites d'un nouveau prouveur ou d'un nouvel algorithme. Derrière différentes structures de données, de façons de coder, de langages, chaque prouveur SAT implante ses propres techniques de recherches, stratégies et heuristiques. La taille de certaines instances (plusieurs dizaines de milliers de variables, plusieurs centaines de milliers de clauses) rendent désormais une mauvaise implantation d'un algorithme (par exemple

*Ce travail s'effectue en collaboration avec João Marques Silva et Ines Lynce du groupe SAT@INESC, Lisbonne, Portugal

¹<http://www.satlive.org/SATCompetition/2003/index.jsp>

en $O(n^2)$ au lieu de $O(n \log n)$) dramatiquement lente. Ainsi, construire, en partant de zéro, un prouveur SAT compétitif par rapport aux prouveurs de référence est une entreprise de plus en plus difficile car les structures de données deviennent de plus en plus sophistiquées. Il est donc de plus en plus compliqué pour chaque chercheur SAT de maintenir à jour son propre prouveur. Par conséquent, une bonne idée peut, faute de temps, passer inaperçue à cause d'un codage inefficace. Idéalement, tous les algorithmes devraient être évalués avec le même degré d'optimisation. Une plate-forme de prouveurs SAT permet de tendre vers cet objectif. C'est l'une des raisons qui nous a poussées à développer la plate-forme OpenSAT. Cette idée a déjà été proposée dans le cadre du projet SIMO [7]. Cependant, à notre connaissance, ce projet n'a pas connu de suite.

OpenSAT poursuit trois buts. Le premier est de faciliter l'intégration de prouveurs SAT dans des applications. Le deuxième est de fournir des implantations de références des techniques et des algorithmes pour résoudre des problèmes autour de SAT. Le dernier est de permettre à chacun de proposer sa propre implantation des spécifications OpenSAT.

Pour y arriver, le développement *open source* semble être le plus approprié et tout spécialement dans un cadre de recherche. Cette approche suscite par essence un travail collaboratif et favorise la diffusion des travaux. Elle permet par ailleurs un développement, une découverte et une résolution des problèmes selon un cycle très court. Cette construction *open source* se fonde sur une variété d'outils de travail collaboratifs via internet accessible sur <http://www.opensat.org>.

Nous pensons que proposer un noyau comportant des structures de données adéquates et des algorithmes de recherche classique peut aider aux développements compétitifs de nouvelles idées. De plus, nous pensons que le partage des efforts de maintenance sur une plate-forme commune et libre peut permettre à des prototypes de recherche de rester compétitifs.

2 Présentation de OpenSAT

La plate-forme OpenSAT est le descendant direct de deux autres projets, JQUEST [14] et JSAT². Elle est développée dans le cadre des actions universitaires intégrées luso-françaises de la conférence des présidents d'université 2002-2003. Elle réunit le CRIL et le groupe SAT de l'institut technique de Lisbonne.

Les deux projets à l'origine d'OpenSAT sont complémentaires et écrits en Java. Le premier, JQUEST, utilise des structures de données et des algorithmes de recherche très élaborés, mais il souffre d'une programmation de style C+ (programmation C avec des structures de données C++). Le second, JSAT, propose une meilleure conception en terme d'objets et de réutilisabilité mais utilise des structures de données simples.

Le choix du langage Java n'est pas anodin. Pour créer une bibliothèque de ce type, un langage objet est nécessaire pour faciliter la réutilisation. Il est clair que ce choix peut avoir un coût. Par exemple, le polymorphisme qui facilite grandement la program-

²<http://cafe.newcastle.edu.au/daniel/JSAT>

mation induit un léger surcoût³. Cependant, ce surcoût est largement compensé par la facilité d'écriture du programme.

Parmi les langages les plus utilisés nous avons le choix entre C++ et Java. Chacun de ces langages a des avantages et des inconvénients. De nombreux prouveurs SAT sont actuellement écrits en C++ pour des raisons d'efficacité. Il s'agit le plus souvent d'une programmation procédurale classique utilisant le sucre syntaxique du C++. Certains de ces prouveurs sont par ailleurs développés pour une architecture particulière : Chaff [16] et ses descendants sont optimisés pour utiliser au maximum le cache de second niveau des processeurs Pentium. De notre point de vue, les optimisations liées à la machine ou au système d'exploitation ne sont pas directement du ressort du chercheur. Donc, l'efficacité du langage ne doit pas être le critère prépondérant. Par ailleurs, idéalement, un prouveur de recherche devrait pouvoir être exécuté sur n'importe quel système. Or, il est parfois difficile de rendre un programme C ou C++ parfaitement portable. De même, certains *bugs* peuvent passer inaperçus tant qu'on ne change pas d'architecture. Par exemple, pendant la compétition SAT2002, qui se déroulait sur des PC Linux, le prouveur Berkmin [10] (écrit en C) a planté sur une centaine d'instances à cause d'un bug qui ne se révèle que sous Linux, le développement de Berkmin étant fait sous Windows et Solaris.

Quand on met en avant la portabilité et la sûreté du langage, Java se révèle un environnement de choix qui permet de ne pas sacrifier outre mesure l'efficacité. Ce langage a également l'avantage d'être maintenant très répandu et de disposer de très bons outils de développement⁴. Par ailleurs, le développement en Java suscite l'intérêt d'un nombre croissant de chercheurs.

2.1 Objectif : deux niveaux d'utilisation

2.1.1 Premier niveau : simple utilisateur

L'idée de base dans OpenSAT est de regarder un prouveur SAT comme une boîte noire à qui l'on donne une instance SAT et qui répond SAT ou UNSAT. On voudrait pouvoir intégrer cette boîte noire dans n'importe quel programme Java, et idéalement pouvoir changer de boîte noire au fur et à mesure des améliorations sur les prouveurs. On se place ici d'un point de vue *utilisateur*.

Jusqu'à présent, il n'y a pas de véritable standardisation autre que le format de représentation des instances SAT sous forme d'entiers signés DIMACS.

A l'heure où les premières applications industrielles basées sur SAT voient le jour (*model checking*, planification [5, 11]), il n'existe pas de définition d'un *composant* SAT, c'est à dire d'une entité logicielle interchangeable fournissant le service SAT. C'est ce que nous essayons de définir dans OpenSAT avec le premier niveau de spécification. Voir par exemple le code Figure 1.

³On pourrait remarquer que l'utilisation de fonctions induit elle aussi un surcoût mais que l'on songe rarement à s'en passer

⁴Voir entre autres <http://www.eclipse.org> et <http://www.junit.org>

```

// org.opensat.Default contains default implementations
// for an easy start up with the framework.
// default simple data structure for representing a CNF
IFormula formula = Default.cnf();
// create a parser for the Dimacs format.
IParser parser = Default.parser();
// get a solver
ISolver solver = Default.solver();
// read the instance
parser.parseInstance("filename.cnf", formula);
try {
    if (solver.solve(formula)) {
        // instance is SAT
    } else {
        // instance is UNSAT
    }
} catch (TIMEOUTException e) {
    // TIMEOUT
}

```

FIG. 1 – Silhouette d’un algorithme SAT avec OpenSAT

2.1.2 Deuxième niveau : chercheur

Un second niveau de spécification permet d’écrire son propre prouveur SAT en utilisant les structures de données existantes pour créer de nouveaux algorithmes, ou de modifier les structures de données sans modifier les algorithmes.

Pour créer ou modifier des structures de données, il suffit d’implanter les interfaces (IClause, ILiteral...) contenues le paquetage `org.opensat`. Vous n’avez alors aucune contrainte de programmation sauf celle de respecter les contrats liés aux interfaces. La mise en place d’une nouvelle structure de données ne requiert aucune modification des algorithmes grâce au polymorphisme.

La création de nouveaux algorithmes s’effectue en réutilisant toutes les structures de données et toutes les classes déjà existantes. La conception objet et la notion d’héritage sont d’un grand secours. En effet, il n’est pas nécessaire de réinventer la roue pour tester une nouvelle idée ou un nouvel algorithme. Le mot clé ici est la réutilisabilité. Il n’y a qu’à se focaliser sur le nouvel algorithme ce qui permet de gagner un temps considérable.

2.2 Une implantation de référence d’algorithmes pour SAT

À terme, OpenSAT doit contenir un ensemble d’algorithmes ayant prouvé leur efficacité pour la résolution du problème SAT. Cela donnera un caractère éducatif au projet, mais permettra également de comparer plus honnêtement ces divers algorithmes.

Les structures de données actuellement implantées sont :

- Implantation classique dite “des compteurs” par tableau ou par liste
- Implantation des "Watched Literals" [16]

Pour l'instant, les seuls prouveurs utilisables sont basés sur la procédure de Davis, Lovemann et Loveland (DPLL) [6]. Ils contiennent les techniques de recherche suivantes :

- Retour arrière intelligent de RELSAT (voir [2]).
- Apprentissage de clauses : plusieurs politiques d'apprentissage peuvent être choisies (taille des clauses, hauteur de l'arbre de recherche...). Il est également très facile de mettre au point sa propre politique d'apprentissage (par simple héritage).
- Traitement local (littéraux purs, propagation des littéraux impliqués ([1, 4]...)).
- Diverses heuristiques de points de choix : Jeroslow et Wang [9], VSIDS [16]. La création de nouvelles heuristiques est également très simple. Il suffit d'implémenter l'interface `IHeuristic`. Les calculs internes aux heuristiques peuvent être faits incrémentalement ou à chaque appel.

La version de référence 0.44, celle que nous avons soumise à la compétition SAT2003, contient le retour arrière non chronologique, l'apprentissage de clauses de taille strictement inférieure à 5, les "watched literals" ainsi que l'heuristique `VSIDSLastMaters`. À court terme, nous avons prévu les extensions suivantes :

- Retour non chronologique comme cela est fait dans GRASP [15] ou dans `zChaff` [16]
- Recherche des symétries [3]
- Prouveur basé sur la recherche locale [8].

2.3 Extensions prévues

La plate-forme OpenSAT offrira à terme des possibilités intéressantes qui sont en général absentes des prouveurs développés : la possibilité d'interrompre une recherche et de la sauvegarder pour la poursuivre ultérieurement, la possibilité de distribuer sur plusieurs ordinateurs l'exploration de l'arbre de recherche ainsi que la visualisation de l'arbre de recherche exploré. Toutes ces extensions se fondent sur une abstraction qui permet de représenter un point de choix.

2.3.1 Points de choix

La notion de point de choix est bien sûr fondamentale dans toute recherche de solution d'un problème SAT. De nombreux algorithmes se fondent sur la procédure DPLL et utilisent des points de choix binaires avec une exploration en profondeur d'abord. Cependant, s'il s'agit là du choix le plus efficace dans la plupart des cas, des points de choix non binaires ou des recherches en largeur d'abord peuvent se révéler plus intéressants dans certaines portions de l'arbre de recherche.

Pour pouvoir utiliser différents types de points de choix au cours de la recherche, il faut bénéficier d'une abstraction permettant de les représenter. Il est clair que l'introduction de cette abstraction aura un coût en termes d'efficacité par comparaison avec des procédures qui coderaient directement ces points de choix. Cependant, le but recherché ici est de faciliter l'exploration de nouveaux algorithmes et les gains de performances qu'ils peuvent apporter. Cette abstraction se révélera d'autant plus intéressante qu'elle rendra très faciles les extensions qui seront détaillées par la suite.

Un extrait de l'interface en cours de conception est le suivant :

```

// un point de choix
public interface IChoicePoint {
    IChoicePoint getParent(); // obtenir le père de ce point de choix

    boolean hasMoreBranches(); // reste-t-il des branches à explorer ?
    IBranch nextBranch(); // si oui, quelle est la prochaine branche ?

    boolean explore(); // explorer toutes les branches et
                       // renvoyer le résultat de la recherche
    ...
}

// une branche d'un point de choix
public interface IBranch {
    void down(); // effectuer les interprétations portées par cette branche
    void up(); // backtrack
    void attach(IChoicePoint cp); // rattacher un nouveau point de choix
    ...
}

```

Cette interface permet d'écrire un prouveur récursif en deux lignes :

```

IChoicePoint racine=new DPLLChoicePoint(formula,heuristic);
boolean satisfiable=racine.explore();

```

L'exploration itérative de l'arbre de recherche est bien sûr également possible.

L'intérêt de cette abstraction est de permettre de gérer uniformément divers types de points de choix. On peut envisager entre autres :

- des points de choix non binaires (par exemple, point de choix spécifique ant comment satisfaire une clause donnée pour la recherche des impliquants premiers)
- un changement de type de point de choix au cours de la recherche
 - pour utiliser une heuristique ou une méthode différente entre le sommet de l'arbre et ses racines.
 - pour des recherches spécifiques (points de choix universels ou existentiels dans le cas des formules QBF)
- pour effectuer d'autres types de parcours dans certaines portions de l'arbre (en largeur d'abord, aléatoire,...)
- de gérer directement le retour arrière intelligent

2.3.2 Interruption et reprise de la recherche

Dans certains cas, on souhaite pouvoir interrompre une recherche de solution pour la reprendre ultérieurement. Malheureusement, la plupart des prouveurs n'offrent pas cette possibilité. Il existe certes des systèmes de checkpointing permettant de sauvegarder un processus mais ces solutions sont en général fortement liées au système utilisé. L'utilisation de points de choix permet de résoudre très facilement le problème puisqu'il suffit de sauvegarder les points de choix existants ainsi que le point de choix actif.

2.3.3 Distribution de la recherche

La résolution de certaines instances nécessite des ressources considérables (en temps ou en espace). La distribution de l'exploration de l'arbre de recherche sur plusieurs ordinateurs permet de pallier ces problèmes :

- en demandant à d'autres machines d'explorer certaines branches (répartition du temps de calcul nécessaire)
- en demandant à d'autres machines de gérer une partie de la formule (répartition de la mémoire nécessaire)

Là encore, une implantation utilisant les points de choix pourrait bénéficier très simplement de ces extensions prévues.

2.3.4 Visualisation des arbres de recherche

Il est possible de sauvegarder une représentation de l'arbre de recherche exploré pour pouvoir l'étudier par la suite sous forme de graphique hypertexte. Cette étude peut servir à résoudre des *bugs* ou à étudier les choix effectués par les heuristiques afin de les améliorer. Un tel visualisateur est en cours d'écriture et l'utilisation des points de choix permettra d'en bénéficier directement.

2.3.5 passerelle avec les CSPs (CHOCO, ABSCON)

Nous connaissons au moins deux projets similaires au notre dans le cadre plus général des problèmes de satisfaction de contraintes : CHOCO [12] et ABSCON [18].

Le projet OpenSAT est similaire dans ses objectifs au projet CHOCO. Outre ceux déjà cités, la clarté des algorithmes, leur facilité d'utilisation, l'utilisation de la plateforme à des fins pédagogiques, etc. sont des objectifs communs. Ils se différencient essentiellement sur le langage utilisé : CHOCO est basé sur un langage de contraintes, CLAIRE. De part son ancienneté, ce projet est plus mature que OpenSAT. Si OpenSAT est adopté par la communauté SAT, on pourrait concevoir des passerelles entre CHOCO et OpenSAT qui pourraient permettre de comparer, et peut être mélanger, les techniques utilisées dans les deux domaines.

ABSCON partage avec OpenSAT son langage de programmation.

3 Comment contribuer à OpenSAT

Dès le départ, nous avons choisi de diffuser le projet de manière *open source*. Chacun peut le récupérer, le modifier et le redistribuer gratuitement. Afin que les utilisateurs venant du milieu industriel puissent en profiter, nous avons opté pour la licence LGPL (Lesser GNU Public Licence) qui permet d'associer du code propriétaire et du code *open source*. Même dans ce cas, toute modification sur le projet *open source* doit être rendue publique. Le choix de la licence LGPL est important car le succès du projet dépend de son adoption par une large catégorie d'utilisateurs de techniques SAT.

3.1 Télécharger et utiliser

La première règle à observer quand l'on annonce un projet open source est de le faire uniquement lorsque les gens peuvent l'utiliser. C'était le premier but. Actuellement la plate-forme contient quelques algorithmes simples et quelques heuristiques. La distribution contient deux archives jar, la plate-forme proprement dite et un ensemble d'outils permettant de gérer les options en ligne de commande⁵.

Une façon très simple d'utiliser OpenSAT consiste à télécharger les archives depuis <http://www.opensat.org> et à les sauver dans le même répertoire. Vous devez alors avoir en possession une machine virtuelle Java récente (Sun J2SE 1.4.0 par exemple) et lancer : `Java -jar opensat.jar <filename>`. Le temps de calcul maximum par défaut est de 30 secondes, mais vous pouvez le modifier avec l'option `-t 120` par exemple. La liste de toutes les options est disponible en utilisant l'option `-h`.

Lors de l'utilisation de OpenSAT, vous pouvez rencontrer des *bugs* ou des comportements étranges. Vous pouvez alors enregistrer un rapport de bug en utilisant OpenSAT Bugzilla⁶ situé à l'adresse <http://www.opensat.org/bugzilla>. Cela permettra aux développeurs de maintenir plus efficacement le projet.

3.2 Participer

L'utilisation de OpenSAT est une voie pour contribuer à sa diffusion et à son essor. Mais il existe d'autres façons plus actives de le faire.

Nous avons développé la première version de OpenSAT en utilisant une approche de programmation extrême : la conception est basée sur les fonctionnalités actuellement implantées et non sur une vue globale. Cela a demandé des efforts constants de refactorisation. Les personnes intéressées peuvent regarder la conception et l'implantation actuelle et nous faire part de leurs avis de façon à améliorer celle-ci. Cela peut nous aider à atteindre l'un des buts : fournir une plate-forme capable d'utiliser diverses techniques (recherche locale, apprentissage...) nécessitant des besoins très disparates. La première difficulté est de trouver une bonne abstraction et de faire le bon compromis entre généralité et efficacité.

Un des aspects éducatifs de OpenSAT est de proposer une implantation des techniques SAT faisant partie de "l'état de l'art" en essayant de respecter le plus possible les pseudo algorithmes publiés. Ainsi, idéalement, chaque algorithme devrait contenir une référence aux articles qui lui sont associés, ainsi qu'une description de ceux-ci. Étant donné que, jusqu'à présent, nous nous sommes concentrés sur l'implantation et la conception, nous avons besoin d'aide pour nous aider à terminer la documentation de la plate-forme.

La conception d'une plate-forme Java pouvant rivaliser avec les meilleurs prouveurs, basés pour la plupart sur les langages C et C++, est difficile. Notez que cela n'est pas impossible. La bibliothèque COLT⁷ qui est une plate-forme scientifique pour Java est plus efficace que certaines bibliothèques fournissant les mêmes fonctionnalités

⁵voir le site web <http://jakarta.apache.org>

⁶<http://www.bugzilla.org>

⁷<http://hoschek.home.cern.ch/hoschek/colt>

et basées sur le C/C++. Les clés pour parvenir à cet objectif sont une bonne conception et de bonnes structures de données. Les personnes ayant une expérience de programmation Java sont donc les bienvenues pour participer activement au projet OpenSAT.

Il existe plusieurs façons de concevoir la contribution de différentes personnes sur un projet *open source* :

- Tout le monde peut enregistrer les changements dans un dépôt CVS
- Seul un groupe de personnes sélectionnées peut enregistrer les changements dans le dépôt CVS. Les autres doivent envoyer leurs modifications à l'un des membres du précédent groupe.

Pour le moment, nous avons choisi la deuxième option. Cette stratégie doit permettre d'assurer la validité des changements (aspects essentiels lorsque l'on parle de prouveurs SAT).

4 Conclusion

Les prouveurs fondés sur la satisfaisabilité de formules propositionnelles utilisent des structures de données de plus en plus évoluées. Nous avons proposé dans cet article la plate-forme OpenSAT qui vise à permettre le partage par la communauté des scientifiques ou des utilisateurs des structures de données avancées et des algorithmes autour de SAT. Pour permettre une grande liberté dans la réutilisation de ces outils, la plate-forme est construite sur un langage objet, largement répandu et portable : Java.

L'objectif d'OpenSAT est double : favoriser l'utilisation de prouveurs SAT et favoriser le développement de nouveaux algorithmes en proposant directement les techniques les plus récentes, telles que les *watched literals*, le retour arrière intelligent, l'apprentissage de clauses ... ainsi que d'autres extensions telles que la sauvegarde d'une recherche de solution et sa reprise ultérieure, une distribution des calculs sur différentes machines. ... Par ailleurs, OpenSAT autorisera aussi une comparaison des algorithmes dans un cadre commun afin de mieux les évaluer.

Ces objectifs sont d'ores et déjà partiellement atteints. Le site <http://www.opensat.org> permet d'obtenir la dernière version du projet, de se tenir informé du développement, de soumettre des rapports de *bugs* et des demandes d'améliorations. Deux prouveurs liés à OpenSAT (JQUEST2 [14] et l'implantation de référence) ont été soumis à la compétition SAT 2003.

Chacun peut bénéficier de l'effort commun investi dans OpenSAT et faire profiter la communauté de ses meilleures techniques en participant au développement *open source* qui a été retenu pour ce projet.

Cette plate-forme se veut ouverte et ne se restreint pas au développement de prouveur SAT. Elle a d'ores et déjà servi au développement d'un prouveur pour les formules QBF (OpenQBF). Des travaux en cours permettront par ailleurs d'y inclure bientôt les représentations par BDD.

La plate-forme OpenSAT vient à peine de naître. Sa croissance nécessitera sans doute quelques évolutions dans sa conception, mais il ne fait aucun doute qu'avec le soutien de tous, elle deviendra tout à fait mature très rapidement. Les bonnes volontés sont les bienvenues...

Références

- [1] G. Audemard, B. Benhamou, and P. Siegel. AVAL : An enumerative method for SAT. In *Proceedings of first international conference on computational logic CL00, Londres*, volume 1861 of *LNCS*, pages 373–383, 2000.
- [2] R. Bayardo, Jr. and R. Schrag. Using CSP look-back techniques to solve real-world SAT instances. In *Proceedings of the 14th National Conference on Artificial Intelligence and 9th Innovative Applications of Artificial Intelligence Conference (AAAI-97/IAAI-97)*, pages 203–208, Menlo Park, July 27–31 1997. AAAI Press.
- [3] B. Benhamou and L. Sais. Tractability through symmetries in propositional calculus. *Journal of Automated Reasoning*, 12(1) :89–102, February 1994.
- [4] D. Le Berre. Exploiting the real power of unit propagation lookahead. In *Proceedings of the Workshop on Theory and Applications of Satisfiability Testing (SAT2001)*, volume 9 of *Electronic Notes in Discrete Mathematics*. Elsevier Science Publishers, 2001.
- [5] A. Biere, A. Cimatti, E. Clarke, and Y. Zhu. Symbolic model checking without BDDs. In *Proceedings of the conference on automated verification*, volume 1579 of *LNCS*, pages 193–207, 1999.
- [6] M. Davis, G. Logemann, and D. Loveland. A machine program for theorem-proving. *Communications of the ACM*, 5(7) :394–397, July 1962.
- [7] A. Tacchella, E. Giunchiglia, M. Narizzano and M. Vardi. Towards an efficient library for sat : a manifesto. In Henry Kautz and Bart Selman, editors, *Electronic Notes in Discrete Mathematics*, volume 9. Elsevier Science Publishers, 2001.
- [8] H. Hoos and T. Stützle. Local search algorithms for SAT : An empirical evaluation. *Journal of Automated Reasoning*, 24(4) :421–481, 2000.
- [9] R. Jeroslow and J. Wang. Solving propositional satisfiability problems. *Annals of Mathematics and Artificial Intelligence*, pages 167–187, 1990.
- [10] E. Goldberg Jr and R. Schrag. Berkmin : A fast and robust sat solver. In *Proceedings of the design and tests in Europe Conference*, pages 142–149, 2002.
- [11] H. Kautz, D. McAllester, and B Selman. Encoding Plans in Propositional Logic. In *Proceedings of the Fifth International Conference on the Principle of Knowledge Representation and Reasoning*, pages 374–384, 1996.
- [12] François Laburthe and Le projet Ocre. Choco : implémentation du noyau d’un système de contraintes. In *Journées Nationales de la Résolution Pratique des problèmes NP-Complets (JNPC) - Marseille*, 2000.
- [13] C. Li and Anbulagan. Heuristics based on unit propagation for satisfiability problems. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI-97)*, pages 366–371, August 23–29 1997.
- [14] I. Lynce and J. Marques-Silva. Efficient data structures for backtrack search SAT solvers. In *Fifth International Symposium on the theory and applications of satisfiability testing*, 2002.

- [15] J. Marques-Silva and K. Sakallah. Grasp : A search algorithm for propositional satisfiability. *IEEE Transactions on Computers*, 48(5) :506–521, 1999.
- [16] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and Sharad Malik. Chaff : Engineering an efficient sat solver. In *Proceedings of 38th Design Automation Conference (DAC01)*, 2001.
- [17] L. Simon, D. Le Berre, and E. Hirsch. The sat2002 competition. Technical report, August 2002. draft version.
- [18] Frédéric Boussemart Sylvain Mechez, Christophe Lecoutre. Abscon : a prototype to solve cps with abstraction. In *Proceedings of the Seventh International Conference on Principles and Practice of Constraint Programming*, 2001.
- [19] H. Zhang and M. Stickel. Implementing the Davis-Putnam method. *Journal of Automated Reasoning*, 24 :277–296, 2000.