Hybridation des méthodes de résolution pour SAT

Olivier Fourdrinoy

Centre de Recherche en Informatique de Lens, CNRS FRE 2499
Université d'Artois
Rue Jean Souvraz SP 18 F-62307 Lens Cedex
fourdrinoy@cril.univ-artois.fr

Résumé : Dans cet article nous traitons du problème SAT en général et de l'hybridation des méthodes de résolution en particulier. Déjà abordée dans différents travaux (9) cette idée consiste souvent à associer méthodes complètes et méthodes incomplètes afin d'optimiser la résolution du problème SAT en tirant partie des deux approches respectives. Nous proposons ici une nouvelle approche utilisant la recherche locale pour ordonner variables et clauses afin de guider un algorithme complet classique dans ses affectations.

Mots-clés : Logique propositionnelle, Méthode complète, Recherche Locale, Hybridation,SAT.

1 Introduction

Contrairement aux algorithmes énumératifs qui parcourent de façon systématique l'espace de recherche, les méthodes basées sur la recherche locale considèrent des « configurations », c'est-à-dire des instanciations complètes de toutes les variables de la formule. Au départ, ce sont des méthodes d'optimisation (on peut par exemple chercher à maximiser le nombre de clauses satisfaites). Sauf adaptation, ces méthodes sont incomplètes, et ne garantissent donc pas l'obtention d'une solution. En particulier, elles ne permettent pas en général de prouver qu'une instance est insatisfiable. Le principe de base des méthodes de recherche locale consiste à se déplacer judicieusement dans l'espace des configurations en améliorant la configuration courante. Ce type d'approche est généralement très efficace pour résoudre des instances satisfiables de grande taille. Nous étudions ici les possibilités de coopération entre la recherche locale et une méthode complète. Différents schémas de coopération sont observés.

Dans cet article, nous utilisons la recherche locale de deux manières. La première, assez classique puisque qu'elle est utilisée pour éventuellement trouver une solution au problème. La seconde est utilisée lorsque aucun modèle n'est trouvé pour ordonner les clauses et les variables en fonction de leurs apparitions dans les clauses falsifiées au cours de la recherche. A cette approche introduite dans (9) nous apportons des améliorations via différentes modifications notamment sur le paramétrage du nombre d'appels

à la recherche locale et sur la recherche locale elle-même.

Dans un premier temps, nous définissons quelques notions de logique propositionnelle avant de présenter brièvement les principales méthodes de résolution du problème SAT. Nous terminons cette introduction par une taxinomie d'hybridations déjà envisagées pour la résolution du problème SAT. Dans une seconde partie nous présentons notre nouvelle approche avant d'analyser quelques résultats significatifs. Enfin nous concluons en apportant les perspectives envisageables.

2 Le problème SAT

Dans cette partie, nous présentons succinctement la logique propositionnelle puis le problème SAT en développant les deux principales méthodes de résolution.

2.1 Définitions préliminaires

Soit $\mathcal{B}$ un langage booléen (i.e. propositionnel) de formules, utilisant les connecteurs usuels($\vee$, $\wedge$, $\neg$, $\Rightarrow$, $\Leftrightarrow$) et un ensemble de variables propositionnelles.

Une *formule CNF* Σ est un ensemble (interprété comme une conjonction) de *clauses*, où une clause est un ensemble (interprété comme une disjonction) de *littéraux*. Un littéral est une variable propositionnelle positive ou négative. On note $\mathcal{V}(\Sigma)$ (resp. $\mathcal{L}(\Sigma)$) l'ensemble des variables (resp. littéraux) apparaissant dans Σ. Une *clause unitaire* est une clause formée d'un unique littéral. Un *littéral unitaire* est l'unique littéral d'une clause unitaire.

Une *interprétation* d'une formule booléenne est une affectation des valeurs $\{vrai, faux\}$ de ces variables. Un *modèle* d'une formule CNF est une interprétation qui satisfait la formule. Le problème SAT revient à décider si une formule CNF admet un modèle ou à prouver qu'elle n'en admet pas.

Rappelons que toute formule propositionnelle admet une forme CNF qui lui est équivalente.

La plupart des démonstrateurs automatiques exploitent le théorème de déduction. Ils utilisent au moins un test de satisfaisabilité pour montrer qu'une information peut être déduite à partir d'une base de connaissances. Le problème SAT s'est révélé important au moins à ce titre. Par ailleurs, il occupe également une place particulière et centrale parmi les problèmes NP-complets. Toutes ces raisons ont fait que, au cours de ces 50 dernières années, les recherches portant sur le problème SAT ont été nombreuses et variées. De nombreux algorithmes ont notamment été proposés pour le résoudre. Ces algorithmes se décomposent principalement en deux groupes : les algorithmes complets et les algorithmes incomplets.

2.2 Les méthodes complètes

Les algorithmes complets répondent à la fois à la satisfiabilité et à l'insatisfaisabilité. On les dit « complets » car si on leur laisse un temps et un espace illimité, ils parcourent *complètement* l'espace de recherche. Il existe deux principaux types d'algorithmes complets. Les algorithmes syntaxiques et les algorithmes sémantiques. Les

premiers s'intéressent à la structure du problème en ne tenant pas compte des valeurs des variables, on prendra ici pour exemple le principe de résolution. Les seconds s'attachent au sens des variables pour tenter de trouver une solution, ils ont pour principal représentant la méthode de (4).

2.2.1 Le principe de résolution

Le principe de résolution est donc un « algorithme syntaxique ». C'est un principe élémentaire mais fondamental en logique. On applique successivement les trois règles suivantes jusqu'à l'arrêt de la procédure :

- règle de résolution
- règle de fusion
- règle de sous-sommation

Si on arrive sur la clause vide, c'est à dire que la clause générée ne contient plus de littéraux on peut conclure qu'il n'y a pas de solution.

Par contre si l'on ne parvient plus à générer de nouvelles clauses et que l'on n'a pas généré de clause vide, c'est qu'il existe au moins une solution.

Le principe de résolution n'est pas l'algorithme complet le plus utilisé du fait que le nombre de clauses à ajouter dans le pire des cas est exponentiel par rapport à la taille initiale du problème.

La méthode présentée dans (5) propose de choisir une à une les variables et à chaque fois de générer toutes les résolvantes liées à cette variable. Ensuite on peut supprimer toutes les clauses contenant cette variable. Au final, soit on trouve la clause vide et on conclut à l'insatisfaisabilité, soit on ne parvient plus à générer de résolvantes et on conclut à la satisfaisabilité de l'instance. Cependant, la génération d'un modèle est quelque peu fastidieuse et cet algorithme a tendance à saturer la mémoire. En effet, en cours d'exécution, le nombre de clauses générées peut vite s'avérer très important.

2.2.2 La procédure de Davis Logemann et Loveland

La méthode développée par Davis, Logemann, et Loveland appelée parfois DPLL ou encore plus abusivement DP (Putnam apparaissant pour des raisons historiques) paraît beaucoup plus naturelle puisqu'il s'agit à priori de parcourir l'ensemble des solutions de manière systématique. Le *Backtrack* et la *propagation unitaire* ainsi que le traitement des littéraux purs [1] sont les composantes essentielles des procédures de type Davis et Putnam.

L'algorithme de Davis et Putnam fonctionne ainsi. Il nettoie à chaque étape l'ensemble de clauses. C'est-à-dire qu'il élimine systématiquement, les clauses unitaires et les clauses où apparaissent un littéraux purs et supprime les occurences de tous les littéraux opposés aux littéraux unitaires traités. Une fois ce nettoyage effectué, et s'il n'a pas donné lieu à la preuve de la consistance ou de l'inconsistance de la formule, l'algorithme poursuit sa recherche dans l'arbre en affectant la variable de son choix. Ce choix est l'un des points importants de l'algorithme de Davis et Putnam. C'est d'ailleurs une

[1] Un littéral est dit pur dans une CNF si son opposé n'apparaît dans aucune clause de la CNF.

Algorithm 1: Algorithme de Davis et Putnam

fonction *DPLL* **return** *boolean*
Data: un ensemble Σ de clauses
Result: true si Σ est consistant, false sinon
begin
  Σ^* =Propagation_Unitaire(Σ);
  if Σ^* *contient la clause vide* **then** return false;
  else
    Σ^+ =Simplification_Littéraux_Purs(Σ^*);
    if $\Sigma^+ == \emptyset$ **then** return true;
    else
      l=Heuristique_de_branchement(Σ^+);
      return $(DPLL(\Sigma^+ \cup \{(l)\}))\|(DPLL(\Sigma^+ \cup \{(\neg l)\}))$;
end

des principales variations entre les différents algorithmes qui s'en sont inspirés. Une fois le choix fait, DP est appelé récursivement avec les deux sous-formules générées.

De nombreuses améliorations de cet algorithme ont été proposées ces dernières années. Parmi elles on notera les *Watched litterals* dans (11) qui offrent une nouvelle structure de données plus adaptée à la propagation unitaire ainsi que les travaux sur les *dépendances fonctionnelles* dans (12) qui utilisent un pré-traitement pour analyser la structure des problèmes posés.

2.3 Les méthodes incomplètes

Les algorithmes incomplets, par définition, ne parcourent pas *complètement* l'arbre de recherche. En général, ils ne répondent pas à la question de l'insatisfaisabilité.

2.3.1 La recherche locale

Une méthode de recherche locale n'effectue pas un parcours systématique de l'espace de recherche. Le plus souvent elle choisit une affectation des variables du problème au hasard et ensuite, si la solution n'est pas trouvée, on flippe (c'est à dire qu'on inverse sa valeur) les variables une à une en suivant une stratégie de réparation. L'algorithme s'arrête lorsqu'une solution est trouvée ou lorsque le temps ou l'espace dédié à l'algorithme est dépassé.

La méthode de choix de la variable à *flipper* caractérise la méthode de recherche locale. L'objectif de ces méthodes est de s'extraire des *minima locaux*. Il s'agit de situations où quelque soit la variable que l'on flippe, le nombre de clauses insatisfaites ne diminue pas. Il est nécessaire de trouver des heuristiques (dites d'échappement) qui permettent de sortir de ce genre de piège.

Pour cet algorithme, il faut dégager différents points stratégiques : tout d'abord dans les entrées, le nombre de réparations maximal n'est pas anodin. Il est parfois préfé-

Algorithm 2: Algorithme de recherche locale

```
fonction RL return boolean
Data: un ensemble Σ de clauses et max_reparations le nombre maximum de réparations autorisées
Result: true si Σ admet un modèle, false si on ne peut conclure
begin
    Générer une configuration initiale I;
    nb_reparations = 0;
    while (nb_reparations < max_reparations) and (I n'est pas un modèle) do
        if une descente est possible then
            Remplacer I par une interprétation voisine permettant la descente;
        else
            Remplacer I par une interprétation voisine selon le critère d'échappement ;
        nb_reparations++;
    return (I est modèle) ;
    %% retourner la valeur du test : "I est un modèle ?"
end
```

rable de recommencer plusieurs fois l'algorithme à zéro plutôt que de s'enliser dans un nombre trop grand de réparations.

Ensuite la génération de la configuration initiale peut se faire de différentes manières. Il est par exemple possible de choisir une répartition 50/50 des littéraux vrais et des littéraux faux, ou encore mettre tout à vrai ou tout à faux. Ces choix peuvent s'avérer important selon la nature du problème.

Enfin, vient le choix de la variable à « flipper » qui s'effectue dans la boucle *while*. Certains algorithmes de recherche locale prônent une analyse totale du problème pour choisir celle qui aura le meilleur rendement. D'autres, pour gagner du temps, préfèrent prendre une variable au hasard. Les meilleures solutions s'appuient sur un compromis entre ces deux extrêmes.(Selman *et al.*; 8)

2.4 Les méthodes hybrides

La notion d'hybridation est ici un abus de langage. Les algorithmes de résolution pour SAT sont complets ou incomplets. La notion d'hybridation est dérivée du fait que l'on greffe généralement entre eux des algorithmes complets et incomplets. Le résultat est un algorithme complet ou incomplet selon la nature de la greffe. Mais en aucun cas il ne s'agit d'un troisième statut.

Dans le cas de SAT, l'hybridation revient d'un coté à utiliser des heuristiques inspirées de DPLL pour la recherche locale et réciproquement des heuristiques inspirées de la recherche locale pour DPLL. Les résultats donnent donc des méthodes complètes ou incomplètes.

La collaboration entre les méthodes systématiques et les méthodes non systématiques constitue l'un des challenges proposés par Selman(14). L'idée de ce challenge est de montrer que la coopération des deux méthodes est plus efficace que chacune d'elles prises séparément.

Pour présenter ces méthodes dites *hybrides*, nous les classons par souche. Par exemple, si on utilise une recherche locale pour trouver la variable de branchement d'une approche systématique, on considère que la souche est de type complète (ou systématique).

2.4.1 Schémas de coopération basés sur une souche incomplète

Nous trouvons ici les méthodes de résolution qui se basent sur une méthode incomplète. Notons que cette approche a surtout été explorée dans le domaine des CSP. Comme il est écrit dans un paragraphe précédent, on peut caractériser une recherche locale par l'ensemble des opérations suivantes :

- le choix d'une assignation initiale des variables ;
- le choix de la prochaine assignation ;
- un critère d'arrêt.

Pour chacune de ces opérations, il est possible d'utiliser des informations issues des méthodes complètes. Nous présentons une liste non-exhaustive de méthodes hybrides basées sur des méthodes incomplètes.

– **résolution alternée :** Zhang & Zhang (16) effectuent une assignation initiale partielle des variables. La recherche continue par un algorithme de *backtrack* (type DPLL) qui tente de résoudre le sous problème induit. Le processus est ensuite réitéré. On parle ici de *résolution alternée* car les méthodes travaillent chacune leur tour.

– **résolution de sous problème par DP :** Lorsque l'on se trouve confronté à un noyau de clauses difficilement satisfiables, on peut tenter de résoudre le sous-problème de manière systématique. Cette approche peut permettre par ailleurs de prouver l'inconsistance ce qui n'est pas le cas d'une méthode incomplète classique. En réalité cette méthode est plutôt exploitée dans l'autre sens, c'est à dire que l'on utilise la recherche locale pour détecter des noyaux inconsistants (2) et y choisir les variables de branchement pour une recherche systématique.

– **recherche locale complète :** Plus récemment(Fang & Ruml), H.Fang et W.Ruml ont proposé un nouvel algorithme du nom de *cls* (pour *Complete Local Search*) qui tout en se basant sur la recherche locale offre la particularité d'être complet. L'algorithme *cls* effectue une recherche locale avec pour heuristique d'échappement l'ajout de clauses en utilisant le principe de résolution. Ainsi, soit la recherche locale donne un modèle, soit on ne peut plus générer de clause et dans ce cas il y a un modèle, soit la clause vide est générée et le problème est déclaré insatisfiable. Typiquement, cette coopération rend la recherche locale complète.

2.4.2 Schémas de coopération basés sur une souche complète

Beaucoup plus prolixe que la précédente, cette coopération cherche à exploiter les nombreuses informations fournies par la recherche locale afin de guider l'algorithme dans le parcours de l'arbre des solutions.

– **Complétude partielle :** méthode duale à la résolution alternée, la *complétude partielle* limite DP à une profondeur variable au delà de laquelle une recherche locale

est lancée. La profondeur dépend d'un ensemble de paramètres tels que le temps disponibles ou l'estimation de la meilleure solution espérée.

– **pré-traiter pour ordonner :** la recherche locale est utilisée pour un pré-traitement du problème. Elle peut fournir éventuellement une solution pour des instances SAT mais l'objectif est avant tout d'identifier les éléments les plus problématiques de l'instance.
  Il existe plusieurs alternatives suivant que l'on s'intéresse aux clauses, aux variables ou encore aux littéraux. Ensuite, la recherche systématique est guidée par les informations collectées durant la recherche locale. Concrètement on effectue un ordonnancement des variables ou des littéraux en fonction de leurs apparitions ou non dans les clauses régulièrement insatisfaites par la recherche locale. De nombreuses approches sont possibles selon que l'on utilise tel ou tel algorithme de recherche locale et suivant l'heuristique de pondération choisie.
  J.Crawford(3) expérimenta cette méthode sur les instances 3SAT aléatoires au seuil sans succès alors que B.Mazure(7) a obtenu des résultats plus probant sur un plus large panel d'instances. Une approche similaire en se basant sur les littéraux permet d'affiner l'ordonnancement des variables.
  Il est encore possible de pondérer les clauses et d'ordonnancer celles-ci. Ensuite DP prendra comme variable de choix les variables de la clause la plus falsifiée et ainsi de suite.
– **Heuristique de branchement :** L'utilisation de la recherche locale comme heuristique de branchement (9) prend en compte le fait que l'ordonnancement peut évoluer au fur et à mesure de l'exploration de l'arbre de recherche. Ainsi, au lieu de se baser sur un ordonnancement statique des variables (des clauses ou des littéraux), on recalcule régulièrement celui-ci. DPRL appelle ainsi à chaque point de choix une heuristique de type recherche locale qui si elle ne trouve pas de solution, identifie le littéral apparaissant le plus souvent dans les clauses falsifiées.
  La fréquence des appels à la recherche locale est bien entendue paramétrable et c'est d'ailleurs l'inconvénient majeur de la méthode. Une recherche locale utilise des ressources de temps non négligeable et un recours systématique à celle-ci peut s'avérer parfois coûteux.

Notre approche s'inspire par ailleurs de cette dernière idée.

3 Description de notre approche

Nous avons travaillé sur une coopération entre la recherche locale et une méthode de type DPLL. Comme nous l'avons vu précédemment, la recherche locale est généralement incapable de prouver l'inconsistance. Elle peut cependant fournir un ensemble d'informations pouvant aider à détecter cette inconsistance. Une grande partie des instances inconsistantes se caractérisent par la présence d'un ou plusieurs noyaux inconsistants. On entend par là que seul un sous-ensemble de variables (et/ou de clauses)intervient dans l'insatisfaisabilité de l'instance. Notre but est de circonscrire cet ensemble par la recherche locale pour ensuite piocher les variables de décision de la méthode systématique dans ce sous-ensemble.

3.1 Détection de noyaux inconsistants

3.1.1 Les noyaux inconsistants

Définition 1
Une CNF Σ *est* **globalement inconsistante** *si et seulement si* Σ *est inconsistante et* $\forall$ $\Pi \subset \Sigma$, Π *est consistante.*

Lorsqu'une CNF inconsistante n'est pas globalement inconsistante, elle est dite localement inconsistante.

Cependant il est possible de définir plusieurs degrés de localité. En effet cette localité peut être caractérisée par le rapport entre la taille de la plus petite sous-CNF inconsistance et la taille de la CNF initiale. Les sous-CNF inconsistantes de la base sont appelées *noyaux inconsistants.*

Définition 2
Soit Σ *une CNF inconsistance.* Π *est appelé* **noyau inconsistant** *de* Σ *si et seulement si* $\Pi \subset \Sigma$ *et* Π *est inconsistant. On dit que* Π *est* **noyau globalement inconsistant** *si et seulement si* Π *est globalement inconsistant. Un noyau inconsistant* Π *de* Σ *est dit* **minimalement inconsistant** *si et seulement si* Π *est globalement inconsistant et pour tout noyau inconsistant* Ω *de* Σ, $|\Pi| \leq |\Omega|$.

Ces différentes formes d'inconsistance présentent un intérêt majeur notamment dans les applications dérivant de SAT. L'inconsistance d'une base de connaissances est souvent due à la présence accidentelle d'une information contradictoire. La détection de ces données contradictoires est utile dans de nombreuses applications, comme par exemple la détection de pannes.

3.2 Recherche locale et détection de noyaux inconsistants

A chaque fois qu'une recherche locale propose une interprétation falsifiée, on sait qu'au moins une clause de chaque noyau inconsistant est présente dans les clauses falsifiées par l'interprétation courante. Cette propriété fournit un ensemble d'informations pouvant amener à détecter des noyaux inconsistants plus rapidement dans une méthode systématique. L'idée majeure est donc d'exploiter au « maximum » les informations fournies par la recherche locale dans le but de circonscrire un noyau inconsistant. En effet, un seul noyau suffit pour montrer l'inconsistance d'une CNF.

Un noyau inconsistant se compose de clauses qui elles-mêmes se composent de littéraux. On peut donc identifier un noyau inconsistant de deux manières :

– en identifiant les clauses.
– en identifiant les littéraux.

Pour ces derniers, l'inconsistance vient bien évidemment de ce que les littéraux sont présents à la fois positivement et négativement. Donc on cherche plutôt à identifier les variables du noyau.

Nous avons donc développé la méthode suivante. Nous avons marqué les clauses les plus falsifiées au cours de la recherche locale. Ainsi, à la fin de celle-ci, si aucun modèle

n'a été détecté, nous avons un ordre sur les clauses qui permet de guider la méthode systématique dans ses choix de variables de décision.

Cependant, la recherche locale se caractérise par une phase de « descente ». En effet, après avoir affecté aléatoirement les variables, on cherche à diminuer le nombre de clauses falsifiées. Ce nombre de clauses fausses initiales peut s'avérer important sans que cela soit représentatif du problème. En général, ce nombre décroît rapidement au cours des premiers *flips* avant de se stabiliser. Les informations relatives à cette descente ne sont pas à priori significatives. On pourrait aisément les assimiler à un phénomène de « bruit » dans le domaine des statistiques. Afin de ne pas perdre de temps sur ces dernières, nous avons développé deux stratégies : une basée sur un seuil l'autre sur les minima locaux.

3.2.1 Heuristique de pondération avec seuil

Définition 3
Le **seuil de pondération** *est une constante entière qui indique le nombre maximum de clauses falsifiées en dessous duquel on marque celles-ci.*

La notion de seuil se base sur le fait que la recherche locale bute en général sur un très petit nombre de clauses. Ainsi, on ne marquera que les clauses (ou les littéraux) présentes dans le cas de figure suivant : le nombre de clauses falsifiées est inférieur au seuil k fixé par l'utilisateur. Toute la difficulté réside ici dans le réglage du seuil. La valeur optimale du seuil est fortement liée à la nature de l'instance testée. Nous avons obtenu des résultats intéressants avec cette stratégie mais un problème très simple s'est posé. Si n noyaux inconsistants indépendants (qui ne possèdent pas de variables en commun) constituent le problème, il faudra que le seuil soit au moins égal à n. Or nous ne sommes pas capables de savoir à l'avance combien de noyaux constituent l'instance. Aussi avons nous développé l'heuristique suivante.

3.2.2 Heuristique de pondération basée sur les minima

En marquant les clauses présentes dans les minima locaux[2], l'algorithme évite de perdre du temps dans les « descentes » et il n'est pas tributaire du nombre de noyaux inconsistants. Cette stratégie semble plus pertinente car elle ne nécessite pas de régler le seuil. Celui-ci fluctue dynamiquement au cours de la recherche locale. Cependant, si l'heuristique d'échappement ne permet pas de se sortir d'un creuset d'insatisfaction, il peut être difficile de cerner le noyau inconsistant.

3.3 Une heuristique de branchement basée sur la recherche locale

L'idée de cette contribution est de proposer une nouvelle combinaison qui utilise la recherche locale comme heuristique de branchement pour un algorithme de type DPLL. Nous utilisons donc une version basique de l'algorithme DPLL et un algorithme de recherche locale WSAT ou « WalkSat » (13).

[2]Un minimum local est une interprétation telle que quelle que soit la variable flippée, il est impossible d'améliorer le nombre de clauses satisfaites.

Il existe encore de nombreuses options possibles quant au paramétrage de cette collaboration :

- utiliser WSAT systématiquement ;
- utiliser WSAT jusqu'à une profondeur prédéfinie puis une heuristique classique ;
- utiliser WSAT pour une profondeur proportionnelle à la taille de l'instance traitée puis une heuristique classique ;

Nous comparons ces trois approches dans la partie 4.

L'algorithme (WSAT) se singularise par sa méthode de choix de la variable à flipper. En effet, au lieu d'utiliser une heuristique qui va parcourir toutes les clauses et donc toutes les variables afin de choisir celle dont le « flip » paraît le plus utile, WSAT prend une clause falsifiée au hasard, et se concentre sur les variables de cette clause. Cette méthode surprenante présente un gain de temps non négligeable. Par exemple, sur une instance 3-SAT, on ne regardera que 3 variables au maximum même si l'instance compte des milliers de variables.

Pour chacune des variables contenues dans la clause falsifiée, on va attribuer un « score » et la variable de plus grand score sera choisie pour être inversée.

La fonction Score fonctionne très simplement en calculant le nombre de clauses qui seront falsifiées en cas de flip de la variable, puis le nombre de clauses qui seront satisfaites par ce même flip et en renvoyant la différence entre ces deux valeurs. On préférera bien évidemment les variables favorisant l'augmentation du nombre totale de clauses satisfaites.

3.4 Intégration de l'heuristique dans DPLL

Après la recherche locale, nous obtenons un tableau contenant les scores des clauses et nous pouvons donc établir un classement des clauses les plus falsifiées. Nous pouvons également établir un classement des variables les plus souvent présentes dans les clauses falsifiées. En utilisant ce second classement, il est possible d'affecter en priorité les variables posant le plus de problème. Cependant, dès qu'une affectation est faite, les clauses les plus difficilement satisfiables sont validées et il peut être intéressant de relancer la recherche locale afin de voire quelles sont dès lors les nouvelles clauses « difficiles ».

3.4.1 Fréquence de l'appel à la recherche locale

Nous avons suite à différentes expérimentations sur les instances aléatoires fixé le paramétrage suivant. La recherche locale est appelée tant que la profondeur de l'arbre d'exploration des solutions est inférieure à 5. Ensuite, on se base sur le dernier classement des variables obtenus pour choisir les variables d'affectations.

3.4.2 Diversification de la recherche locale

Lors de nos premiers tests, nous choisissions l'interprétation initiale au hasard (en prenant soin de ne pas toucher aux variables déjà affectées dans DPLL). Une amélioration consiste à construire la nouvelle interprétation initiale à partir des dernières affectations connues des variables à affecter. En effet, ces affectations résultant soit d'un

backtrack dans DPLL, soit de la dernière recherche locale, cela permet de se concentrer sur des affectations apparemment défectueuses. Puis afin de diversifier les terrains d'explorations de la recherche locale, nous effectuons un flip général sur toutes les variables (concernées par le recherche locale). En somme, la recherche locale ressasse les derniers problèmes rencontrés puis envisage les configurations opposées.

L'idée est que nous n'utilisons pas la recherche locale dans son but premier (trouver une solution). Aussi nous souhaitons explorer le maximum d'interprétations différentes et cette méthode prétend diversifier les recherches.

3.4.3 Synthèse

- tant que la profondeur de l'arbre est inférieure à 5 :
  - WSAT + heuristique de pondération basée sur les minima locaux ;
  - affectation et propagation de la variable de meilleur score ;
- affectation et propagation des variables en suivant le dernier classement fournit par WSAT.

4 Résultat expérimentaux et analyses

Nos expérimentations ont eu pour but de valider expérimentalement notre approche en la comparant à des approches plus classiques.

Pour ces expérimentations, nous avons utilisé les 4 prouveurs suivants :

- un DPLL avec 1 appel à la recherche locale appelé DP_1_RL (9) ;
- une recherche locale RL : WSAT ;
- un DPLL avec accès systématique à la recherche locale DP_ALL (9) ;
- notre prouveur décrit précédemment.

Nous avons comparé les approches sur deux types d'instances : les instances aléatoires issues du modèle classique de génération et les instances AIM issues de DIMACS(1)

Les expérimentations ont été réalisées sur des P3 fonctionnant sous linux Fedora core 2.

Les résultats détaillés en annexe correspondent au temps moyen en seconde mis pour résoudre les instances. Un TIME OUT de 1000 secondes a cependant été fixé.

Les résultats expérimentaux montrent que les utilisations unique, ou au contraire systématique de la recherche locale comme heuristique de branchement ont de moins bon résultats que l'utilisation partielle de celle-ci. Il est évident que la recherche locale utilisée systématiquement coûte très cher, surtout sur les instances insatisfiables. De plus l'utilisation unique (au lancement) de la recherche locale patit d'un défaut majeur de la recherche locale. En effet, celle-ci a tendance à explorer des parties relativement petites de l'espace de recherche. On risque donc de baser DPLL sur un cas particulier de l'espace de recherche, d'autant que pour simplifier notre paramétrage, nous avons fixé le nombre de flips.

Les résultats obtenus sont conformes à nos attentes. Les différentes approches présentent des résultats cohérents. En effet la recherche locale est plus efficace que DP sur les instances SAT.

instances	DP_1_RL	RL	DP_ALL	notre prouveur
SAT	145.13	111.37	153.31	85.27
UNSAT	306.48	#	325.7	136.41

TAB. 1 – tableau de synthèse

DP_1 et DP_ALL montrent certaines limites. On peut supposer que DP_1 paye le manque de réactivité de son heuristique. En effet une fois la recherche locale terminée, il utilise constamment le classement fourni et ne met jamais à jour les données qu'il contient. Par contre DP_ALL semble pâtir du trop grand nombre d'appels à la recherche locale.

Un appel mesuré à la recherche locale semble porter ses fruits sur les instances UNSAT, ce que nous espérions mais également sur les instances SAT où notre approche surpasse la recherche locale. Nous pensons que ces résultats sont dus au fait que notre heuristique permet alors de cerner non pas les noyaux inconsistants, mais ce qu'on pourrait appeler les noyaux de difficulté. Des sous-ensembles de clauses présentant un petit nombre d'interprétations possibles.

5 Conclusion et perspectives

Dans ce papier nous apportons une nouvelle approche de coopération entre la recherche locale et les méthodes systématiques de résolution pour SAT. Les résultats présentés montrent un comportement intéressant même s'ils se limitent à des algorithmes très basiques. La lourdeur du paramétrage de telles collaborations nous à amener à faire des choix arbitraires et à utiliser en premier lieu de tels algorithmes. La pertinence de ces résultats nous encourage à porter ce travail vers des solveurs plus aboutis.

Il pourrait également être intéressant d'explorer les notions de « heavy-tail » (6), de « backbone » (10) ou encore de « backdoor » (15) qui constituent à l'instar des noyaux inconsistants d'autres façons de caractériser le "noyau dur" d'un problème.

Références

(1993). : Center for Discrete Mathematics and Computer Science of Rutgers University.

CHVTAL V. & SZEMEREDI E. (1988). Many hard examples for resolution. *Journal of the ACM*.

CRAWFORD & AUTON (1993). Experimental results on the crossover point in satisfiability problems. In *Proceedings of the Eleventh National Conference on Artificial Intelligence (AAAI'93)*, p. 21–27.

DAVIS, LOGEMANN & LOVELAND (1962). A machine program for theorem-proving. *Journal of the Association for Computing Machinery*, **5**, 394–397.

DAVIS & PUTNAM (1960). A computing procedure for quantification theory. *Journal of the Association for Computing Machinery*, (7), 201–215.

FANG & RUML. Complete local search for propositional satisfiability.
GOMES C. P., SELMAN B., CRATO N. & KAUTZ H. (2000). Heavy-tail phenomena in satisfiability and constraint satisfaction. *Journal of Automated Reasoning*.
MAZURE (1999). *De la satisfaisabilité à la compilation de bases de connaissances propositionnelles*. PhD thesis, université d'artois.
MAZURE, SAÏS & GRÉGOIRE (1997). Tabu search for sat. In *Fourteenth National Conferece on Artificial Intelligence*, p. 281–285, Providence(Rhode Island, USA).
MAZURE, SAÏS & GRÉGOIRE (1998). Boosting complete techniques thanks to local search. *Annals of Mathematics and Artificial Intelligence*, **22**, 309–322.
MONASSON R., ZECCHINA R., KIRKPATRICK S., SELMAN B. & TROYANSKY L. (1999). Determining computational complexity from characteristic 'phase transitions'. *Nature*.
MOSKEWICZ, CONOR, MADIGAN, ZHAO, ZHANG & MALIK (2001). Chaff : Engineering an Efficient SAT Solver. In *Proceedings of the 38th Design Automation Conference (DAC'01)*.
OSTROWSKI, GRÉGOIRE, MAZURE & SAIS (2002). Recovering and exploiting structural knowledge from cnf formulas. In *Eighth International Conference on Principles and Practice of Constraint Programming (CP'2002)*, p. 185–199, Ithaca (N.Y.).
SELMAN, KAUTZ & COHEN (1994). Noise strategies for improving local search. In *Twelfth National Conference on Artificial Intelligence(AAAI'94)*, p. 337–343.
SELMAN, KAUTZ & MCALLSTER (1997). Computational challenges in propositional reasoning and search. In *Fifteenth International Joint Conference on Artificial Intelligence(IJCAI'97)*, p. 50–54, Nagoya(Japan).
SELMAN, LEVESQUE & MITCHELL. Gsat : A new method for solving hard satisfiability problems. In *the tenth National Conference on Artificial Intelligence*, p. 440–446 : AAAI'92.
WILLIAMS R., GOMES C. P. & SELMAN B. (2003). Backdoors to typical case complexity. In *Proc. of IJCAI'03*.
ZHANG & ZHANG (1995). Sem : a system for enumerating models. p. 298–303.

Annexe

instances	DP_1_RL	RL	DP_ALL	notre prouveur
aim_50_yes	0	0.39	0.05	0.01
aim_50_no	0.02	#	0.11	0.06
aim_100_yes	126.51	13.28	140.89	0.11
aim_100_no	529.28	#	750.1	0.24
aim_200_yes	485.04	516.51	569.13	502.17
aim_200_no	1000	#	1000	557.39

TAB. 2 – résultats comparatifs sur les AIM

instances	nbVar	nbCla	DP_1_RL	RL	DP_ALL	notre prouveur
al325	50	163	0.1	0.14	0.1	0.12
al325	100	325	0.1	0.13	0.1	0.14
al325	150	488	0.1	0.15	0.1	0.14
al325	200	650	0.09	0.14	0.1	0.12
al325	250	813	0.25	0.14	0.13	0.15
al325	300	975	0.11	0.17	0.4	0.15
al425s	50	213	0.01	0.07	0.01	0.01
al425s	100	425	0.03	4.81	0.04	0.55
al425s	150	638	1.25	82.18	1.22	2.19
al425s	200	850	41.48	243.21	100.86	6.33
al425s	250	1063	656.38	270.36	644.33	79.5
al425s	300	1275	865.45	538.94	887.23	687.31
al425u	50	213	0.01	#	0.01	0.07
al425u	100	425	0.08	#	0.13	1.29
al425u	150	638	1.71	#	4.77	5.63
al425u	200	850	102.75	#	276.16	42.86
al425u	250	1063	937.22	#	1000	239.95
al425u	300	1275	1000	#	1000	949.38
al525	50	263	0.01	#	0	0.03
al525	100	525	0.05	#	0.03	0.44
al525	150	788	0.31	#	0.42	3.28
al525	200	1050	5.55	#	10.43	17.46
al525	250	1313	92.56	#	204.41	61.97
al525	300	1575	867.64	#	965.84	166.1

TAB. 3 – résultats comparatifs sur les aléatoires