A propos des Régles de Branchement pour les Problèmes 3-SAT Aléatoires Insatisfiables et Difficiles

Chu min Li
LaRIA,
5 rue du moulin neuf
80000 Amiens
email : `cli@laria.u-picardie.fr`

Gérard Sylvain
LaRIA,
5 rue du moulin neuf
80000 Amiens
email : `gerard@laria.u-picardie.fr`

Résumé

Nous étudions, pour les instances 3-SAT aléatoires insatisfiables et difficiles, la limite de la règle de branchement dans la procédure DP. Nous essayons pour cela de définir ce que serait la taille minimum de l'arbre de recherche si, à chaque noeud, nous branchions sur la meilleure variable. L'intérêt réside dans la quantité de travail liée à la recherche de règle de branchement optimale. Notre expérimentation nous montre que l'état actuel des recherches, dans le cas des instances SAT insatisfiables et difficiles, est déjà très proche de cet optimum. Plus particulièrement, nous pouvons probablement affirmé que le premier des dix challenges formulé par Selman et al. dans [15], ayant pour nom : prouver qu'une instance 3-SAT aléatoire difficile et insatisfiable à 700 variables, ne pourra pas, à moins qu'une méthode significativement différente du branchement soit trouvée, être résolue par une procédure DP.

1 Introduction

Considérons, sur un ensemble de variables, une formule propositionnelle $\mathcal{F}$ sous Forme Normale Conjonctive (CNF). Le problème de *Satisfiabilié* consiste à tester si l'ensemble des clauses de ($\mathcal{F}$) peuvent être simultanément satisfaites par un assignement de valeurs de vérité (0 ou 1) aux variables. Si chaque clause contient exactement r littéraux ce sous-problème est appelé r-SAT. 3-SAT est le plus petit sous-problème NP-complet de la classe SAT.

SAT est fondamental dans beaucoup de champs de recherche en informatique, électronique et mathématique. La procédure DP [4] est la meilleure des méthodes complètes pour résoudre SAT, celle-ci est décrite dans la figure 1.

La procédure DP construit, tant que l'on ne trouve pas un assignement correct, un arbre binaire de recherche sur l'ensemble des assignements de variables possibles. Si aucune solution n'est trouvée, le problème est insatisfiable. Chaque appel récursif forme un noeud. Il est connu que la taille de l'arbre est généralement exponentielle. Un bon choix de variable est crucial pour la taille du sous-arbre enraciné. En effet, un mauvais choix peut augmenter de façon exponentielle la taille du sous-arbre enraciné.

Beaucoup de recherches se sont focalisées sur la définition d'une heuristique permettant de sélectionner la meilleure variable de branchement. Dans ce papier nous approximons de façon empirique la taille minimum de l'arbre de recherche pour les instances 3-SAT aléatoires insatisfiables et difficiles. L'arbre est minimum si pour chaque noeud la meilleure variable est sélectionnée. Nous nous limitons au cas des instances insatisfiables. En effet, contrairement aux instances satisfiables, le fait de choisir un assignement, pour une variable libre, plutôt qu'un autre dans une instance insatisfiable ne permet pas de réduire la taille de l'arbre à une simple chaîne.

Procédure DP($\mathcal{F}$)
Début
Si $\mathcal{F}$ est vide , Retourner "satisfiable";
$\mathcal{F}$:=UnitPropagation($\mathcal{F}$);
Si $\mathcal{F}$ contient une clause vide, Retourner "insatisfiable";

/* règle de branchement */
Sélectionner une variable x de $\mathcal{F}$ en accord avec l'heuristique H;
Si l'appel de DP($\mathcal{F} \cup \{x\}$) retourne "satisfiable", retourner "satisfiable";
Sinon retourner le résultat de l'appel de DP($\mathcal{F}$
$\cup \{\bar{x}\}$);
Fin

Procédure UnitPropagation($\mathcal{F}$)
Début
Tant qu'il n'y a pas de clause vide et qu'une clause unitaire l existe dans $\mathcal{F}$, l est satisfaite et $\mathcal{F}$ simplifiée;
Retourner $\mathcal{F}$;
Fin

FIG. 1 – *Procédure DP*

Nous étudions les limites des règles de branchement pour les instances 3-SAT aléatoires insatisfiables et difficiles, puisqu'actuellement, celle-ci est la seule technique efficace pour résoudre ce problème. Puisque notre principal objectif est d'étudier cette limite, il est naturel de se focaliser sur l'impact de ces règles.

Les expériences ont nécessité 5 mois de calculs sur 6 PC dotés d'un processeur pentium à 300Mhz sous le système Linux. Nos résultats suggèrent qu'une règle de branchement optimale dans une procédure DP ne permettrait certainement pas d'obtenir de gains significatifs par rapport à l'état de l'art actuel. Plus particulièrement, nous pensons que le premier des dix challenges formulé par Selman et al. in [15], ayant pour nom : prouver qu'une instance 3-SAT aléatoire difficile et insatisfiable à 700 variables, ne pourra pas, à moins qu'une méthode significativement différente du branchement soit trouvée, être résolue par une procédure DP.

Le papier est organisé de la façon suivante. La section 2 présente Satz : la plus rapide, à notre connaissance, procédure DP sur les instances 3-SAT aléatoires. La section 3 présente notre approche, basée sur Satz, permettant d'approximer, pour les instances 3-SAT aléatoires insatisfiables et difficiles, la taille minimale de l'arbre de recherche de la procédure DP. Les résultats expérimentaux y sont également montrés. Dans la section 4, nous essayons de valider notre approche en comparant la valeur approximative avec la taille réelle de l'arbre de recherche minimal. On effectue ce dernier calcul uniquement pour de petites instances 3-SAT aléatoires insatisfiables et difficiles. La section 5 discute des travaux relatifs et la section 6 conclue.

2 Satz

Satz est une procédure DP avec une règle de branchement efficace. Celle-ci sélectionne la variable qui génère le plus de contraintes. La stratégie est basée sur la motivation suivante. Toutes les feuilles (excepté une pour les instances satisfiables) d'un arbre de recherche représentent une impasse où une clause vide a été trouvée. Ainsi pour minimiser un arbre de recherche, il faut essayer d'arriver aux impasses le plus rapidement possible.

Soit $w(l)$, il représente la contrainte introduite dans $\mathcal{F}$ lorsque l est satisfait. $w(l)$ est approximativement égale au nombre de nouvelles clauses de longueur 2 suite à l'exécution de UnitPropagation($\mathcal{F} \cup \{l\}$). Satz branche sur la variable x qui maximise l'équation $w(\bar{x}) * w(x) * 1024 + w(\bar{x}) + w(x)$. La règle de branchement est "two-sided",i.e $w(\bar{x})$ et $w(x)$ sont équilibrés pour une variable de branchement x. Deux exceptions sont traitées dans Satz :

1. si $w(l)$ contient une clause vide, la procédure DP branche sur l, pour ensuite faire un retour-arrière et satisfaire $\bar{l}$.
2. si $w(l) \gg w(\bar{l})$, l n'est généralement pas choisi comme variable de branchement car $w(l)$ et $w(\bar{l})$ ne sont pas équilibrés. Cependant, si $w(l)$ est très élevé, $\mathcal{F}$ devrait posséder d'importantes contraintes suite au branchement sur l. Ces contraintes impliquent probablement une contradiction dans $\mathcal{F}$. Si cette contradiction peut être découverte simplement par une propagation unitaire supplémentaire, la procédure DP branche sur l pour ensuite faire un retour-arrière et satisfaire $\bar{l}$.

Les deux cas ne sont pas traités comme des points de branchement, mais comme une simplification de $\mathcal{F}$ qui satisfait directement $\bar{l}$. Ceci permet de réduire considérablement la taille de l'arbre de recherche. En effet l'arbre de $\mathcal{F} \cup \{\bar{l}\}$ est au plus aussi large que celui de $\mathcal{F}$. Par commodité, nous appellerons, dans ce papier, cette sous-procédure simplification de Satz $FurtherSimplification(\mathcal{F})$. Cette dernière est utilisée dans notre approche puisque des formules $\mathcal{F}$ seront simplifiées avant branchement. Pour plus de détails sur la règle de branchement de Satz, voir [11, 12].

La version originale de Satz présentée dans [12] était déjà la procédure la plus efficace pour les instances aléatoires 3-SAT insatisfiables et difficiles. Rappelons qu'une instance aléatoire 3-SAT est difficile si $m/n \approx 4.25$ [14, 3] où, m représente le nombre de clauses, et n le nombre de variables. Le tableau 1, issue de [12], compare, sur les instances 3-SAT aléatoires (satisfiables incluses), Satz avec trois autres procédures DP issues de l'état de l'art : C-SAT [5], Tableau [3], et POSIT [6].

Des efforts constants sont réalisés afin d'améliorer Satz. Le tableau 2, extrait de [11], nous montre les performances de la nouvelle version de Satz en comparaison avec une version déjà postérieure à celle du tableau 1.

En considérant les efforts consentis pour définir la meilleure règle de branchement, notre question est maintenant de savoir si Satz peut être encore substantiellement amélioré

TAB. 1 – *Temps d'exécution (en seconde sur une station de travail SUN Sparc 20 dotée d'un processeur cadencé à 125MHz) et nombre de retour-arrières (t_size) pour C-SAT, Tableau, Posit et Satz dans le cas d'instances 3-SAT aléatoires difficiles ($m/n = 4.25$)*

	300 vars 300 problems		350 vars 250 problems		400 vars 100 problems	
System	*time*	*t_size*	*time*	*t_size*	*time*	*t_size*
C-SAT	77	49567	512	275303	3818	1624869
Tableau	79	43041	558	253366	4544	1524551
Posit	57	61797	474	400588	3592	2751611
Satz	34	32780	203	174337	1207	916569

TAB. 2 – *Temps d'exécution (en seconde sur un Macintosh G3 300 Mhz avec le système PPCLinux) et nombre de retour-arrières (t_size) pour Satz et sa version améliorée dans le cas d'instances 3-SAT aléatoires difficiles ($m/n = 4.25$). Le gain de la nouvelle version est donné à la dernière ligne*

	350 vars 500 problems		400 vars 300 problems		450 vars 200 problems	
	time	*t_size*	*time*	*t_size*	*time*	*t_size*
Satz	77.64	119248	486	636526	2631	2993061
New Satz	52.02	39908	304	207822	1550	918533
gain	49%	199%	60%	206%	70%	226%

par une simple modification de sa règle de branchement. Ou de façon plus générale, est ce que l'arbre de recherche obtenu en branchant à chaque noeud sur la meilleure variable possible est substantiellement plus petit que celui obtenu par Satz? Nous allons essayer de répondre dans la prochaine section à ces questions pour les instances 3-SAT aléatoires insatisfiables et difficiles.

3 Approximation de la taille minimale de l'arbre de recherche pour les instances 3-SAT aléatoires insatisfiables et difficiles

A un point de branchement, nous comparons toutes les variables de branchement possibles en exécutant Satz sur chacune d'elles. Après le premier choix forcé d'une variable , i.e. sous le point de branchement, Satz travaille de façon normale. Nous collectons ainsi les k meilleures variables afin de brancher sur chacune d'elles. La même procédure est appelée de façon récursive pour les deux fils du point de branchement. La plus petite taille des k sous-arbres enracinés à ce point de branchement est retournée. Nous nommons cette procédure *ApproximateMinSize*, elle est décrite dans la figure 1. Notons que la même sous-procédure de simplification, FurtherSimplification, contenue dans Satz est utilisée pour simplifier $\mathcal{F}$ avant le branchement. Ceci réduit, par fixation de variables, considérablement la taille de l'arbre construit.

```
Procédure ApproximateMinSize(𝓕)
Début
min:=-1; 𝓕:=UnitPropagation(𝓕);
𝓕:=FurtherSimplification(𝓕);
Si 𝓕 contient une clause vide, Retourner 0;
Pour toutes les variables libres x de 𝓕
   Exécuter Satz(𝓕 ∪ {x}) et Satz(𝓕 ∪ {x̄});
Définir S comme l'ensemble des k meilleures variables pour Satz
Pour toutes les variables x de S faire
   début
      size_pos:=ApproximateMinSize(𝓕 ∪ {x});
      size_neg:=ApproximateMinSize(𝓕 ∪ {x̄});
      Si ((min=-1) ou (min > size_neg + size_pos + 1))
         min := size_neg + size_pos + 1;
   fin
Retourner min;
Fin
```

FIG. 2 – *Approximation de la taille minimale pour l'arbre de recherche*

Lorsque k est égal au nombre de variables libres, nous avons exactement le sous-arbre minimum enraciné, puisque la procédure compare tous les arbres possibles. Les arbres de recherche dont les variables de branchement à un point de branchement ne sont pas comprises dans l'ensemble des k variables sont eliminés. Puisque les k variables sont seléctionnées en comparant les arbres de recherche de Satz enracinés à un point de branchement, nous pensons que les arbres de recherche eliminés sont probablement plus grands. De plus, le phénomène "heavy-tailed distribution" identique à celui combattu dans [7] est evité.

Nous effectuons, en variant la valeur de k, trois expériences. Plus k est élevé plus nous obtenons une approximation fine. Malheureusement le temps de calcul devient trop important. Nous allons étudier dans un premier temps le cas $k = 1$. Pour les cas $k \geq 1$, nous nous concentrons sur la racine ainsi que ses deux fils puisque c'est à cet endroit que les variables de branchement possèdent le plus grand impact sur la taille de l'arbre.

3.1 Approximation de l'arbre minimal de recherche dans le cas $k = 1$

Lorsque $k = 1$, ApproximateMinSize devient une procédure DP utilisant Satz pour évaluer l'impact du branchement sur une variable x. A chaque noeud la variable choisie est telle que la somme des tailles des deux sous-arbres, Satz($\mathcal{F} \cup \{x\}$) et Satz($\mathcal{F} \cup \{\bar{x}\}$), soit la plus petite possible. L'arbre obtenu est manifestement plus petit que celui de Satz. Nous exécutons la procédure ApproximateMinSize ($k = 1$) sur des instances 3-SAT aléatoires insatisfiables et difficiles. Le rapport m/n est de 4.25. Nous faisons varier n de 100 à 300 par incrémentation de 20. Pour chaque n nous résolvons 300 instances, et calculons la moyenne des tailles d'arbres obtenus. La figure 3 nous montre la comparaison entre la taille de l'arbre issue d'ApproximateMinSize ($k = 1$) et celle obtenue par la nouvelle version de Satz.

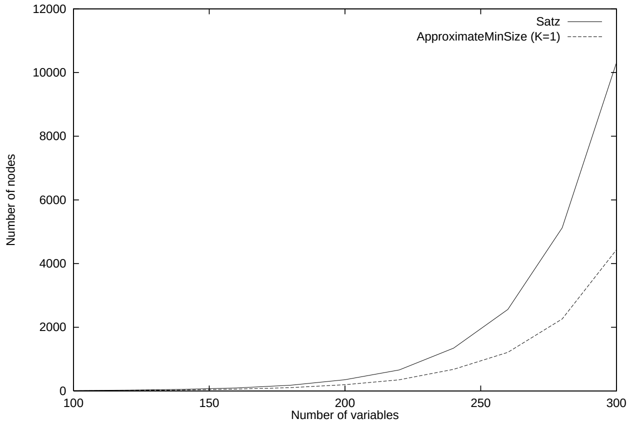

FIG. 3 – *Approximation de la taille minimale pour l'arbre de recherche, et taille de l'arbre de recherche de Satz*

Nous ne remarquons pas de différence substantielle entre Satz et ApproximateMinSize ($k = 1$). Les complexités en taille de l'arbre sont $O(2^{(n/21.14)-1})$ pour Satz et $O(2^{(n/22.18)-1.4})$ pour ApproximateMinSize. Pour les instances insatisfiables à 300 variables la taille de l'arbre de ApproximateMinSize est deux fois inférieure à celui de Satz.

Si nous étendons ces résultats au cas des instances 3-SAT aléatoires insatisfiables et difficiles à 700 variables, la taille de l'arbre de Satz serait de l'ordre de $O(2^{32.11})$, celle de ApproximateMinSize serait $O(2^{30.16})$. En d'autres termes, l'arbre d'ApproximateMinSize serait seulement 4 fois plus petit que celui de Satz.

Nous exécutons Satz sur une telle formule à 700 variables sur laquelle W-SAT [16] n'a pas trouvé de solutions. Satz n'a pas retourné de résultats après 100 heures de calculs, sur un PC sous Linux possédant un processeur pentium cadencé à 300Mhz. Une procédure DP avec la même complexité en taille d'arbre qu'ApproximateMinSize est probablement incapable de résoudre cette formule en un temps raisonnable.

3.2 $k \geq 1$: Arguments en faveur de la procédure ApproximateMinSize

Le fait que l'arbre construit par Satz($\mathcal{F} \cup \{x_1\}$) soit plus petit que celui construit par Satz($\mathcal{F} \cup x_2$) n'assure pas le fait que l'arbre minimum de recherche pour $\mathcal{F} \cup \{x_1\}$ soit plus petit que celui pour $\mathcal{F} \cup \{x_2\}$. Cependant, nous pensons que cela est très souvent le cas pour les instances 3-SAT aléatoires insatisfiables et difficiles. De plus, dans le cas où l'on a réellement une taille d'arbre minimale inférieure pour $\mathcal{F} \cup \{x_2\}$ nous pensons que la différence avec la taille de l'arbre minimal de $\mathcal{F} \cup \{x_1\}$ est négligeable. Nous allons ci-dessous apporter des arguments confirmant notre pensée.

Nous exécutons ApproximateMinSize ($k \geq 1$) afin de prendre en compte les arbres possédant des variables de branchement x dont la somme des tailles des deux sous-arbres obtenus par Satz($\mathcal{F} \cup \{x\}$) et Satz($\mathcal{F} \cup \{\bar{x}\}$) n'est pas minimale. Nous étudions deux cas : (i) $k = 4$ à la racine et $k = 1$ aux autres noeuds; (ii) $k = 2$ à la racine ainsi qu'à ses deux fils, $k = 1$ aux autres noeuds. Notons que dans le cas (i), 4 arbres sont comparés et dans le cas (ii) huit arbres sont comparés.

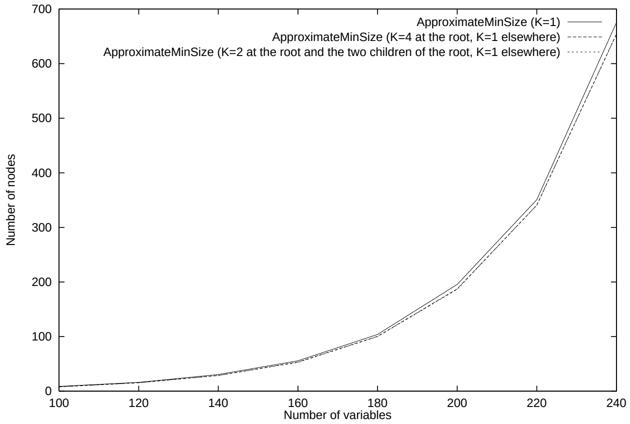

FIG. 4 – *Approximation de la taille minimale pour l'arbre de recherche dans les cas $k = 1$ partout, $k = 4$ à la racine et 1 ailleurs, $k = 2$ aux deux premiers niveaux de l'arbre.*

Nous exécutons ApproximateMinSize ($k \geq 1$) dans les deux cas sur les mêmes instances que $k = 1$. Ceci dit, on se limite à $n = 240$. La figure 4 compare la taille des arbres obtenus par ApproximateMinSize dans les trois cas : $k = 1$; $k = 4$ à la racine; $k = 2$ à la racine et à ses deux fils.

Il est évident que les cas $k \geq 1$ donnent de meilleurs résultat. Ceci-dit, la différence n'est pas significative entre les cas $k \geq 1$ et le cas $k = 1$. En effet, la complexité en taille d'arbre des cas $k \geq 1$ ($O(2^{(n/22.18)-1.5})$) est très proche du cas $k = 1$ ($O(2^{(n/22.18)-1.4})$).

La figure 4 nous confirme que le fait de choisir une variable différente de celle définie comme idéale par Satz ne permettrait certainement pas de réduire de façon significative la taille de l'arbre de recherche. Par conséquent, la taille obtenue par la procédure ApproximateMinSize ($k = 1$) est probablement très proche de la taille de l'arbre réel minimal pour le cas des instances 3-SAT aléatoires insatisfiables et difficiles.

4 Autres Arguments en faveur de la procédure ApproximateMinSize

Il a été prouvé que le problème qui consiste à trouver la meilleure variable de branchement pour un noeud de l'arbre de recherche était à la fois NP-Difficile et Co-NP-Difficile [13]. Par conséquent il est probablement impossible d'obtenir exactement l'arbre minimum de recherche pour des problèmes de taille normale. Cependant, nous pouvons estimer expérimentalement cette taille pour des problèmes 3-SAT aléatoires insatisfiables et difficiles de petite taille. Ainsi il nous est possible de montrer que notre approximation de la taille de l'arbre minimal est très proche de l'optimal.

```
Procédure GetMinSize(𝓕)
Début
   Retourner OptimizedGetMinSize(𝓕,
                                 ApproximateMinSize(𝓕));
Fin

Procédure OptimizedGetMinSize(𝓕,MinSize)
Début
𝓕:=UnitPropagation(𝓕);
𝓕:=FurtherSimplification(𝓕);
Si 𝓕 contient une clause vide, Retourner 0;
Si (MinSize ≤ 0) retourner -1;
ApproxMin := ApproximateMinSize(𝓕);
Si (ApproxMin < MinSize) MinSize := ApproxMin;
Pour toutes les variables libres x de 𝓕 faire
   début
      size_pos:=OptimizedGetMinSize(𝓕 ∪ {x},
                                    MinSize − 1);
      Si (size_pos <> −1)
         size_neg:=OptimizedGetMinSize(𝓕 ∪ {x̄},
                                       Minsize − size_pos − 1);
      Si ((size_pos <> −1 et size_neg <> −1)
          and (MinSize > size_neg + size_pos + 1))
         MinSize := size_neg + size_pos + 1;
   fin
Retourner MinSize;
Fin
```

FIG. 5 – *Calcul de la taille réelle pour l'arbre minimal*

Soit $\mathcal{F}$ une formule SAT, la procédure $GetMinSize$ décrite dans la figure 5 compare tous les arbres possibles et retourne la taille de l'arbre minimal. Pour faire une bonne comparaison avec Satz et ApproximateMinSize, nous utilisons la même sous-procédure FurtherSimplification($\mathcal{F}$). Cette dernière nous permet de simplifier $\mathcal{F}$ en fixant certaines variables libres par propagations unitaires expérimentales. L'arbre de recherche est ainsi considérablement réduit. La procédure GetMinSize peut ainsi traiter des problèmes de plus grandes tailles.

Le paramètre $Minsize$ de la procédure OptimizedGetMinSize représente la taille maximale autorisée de l'arbre créé par une procédure DP pour résoudre une instance $\mathcal{F}$. La construction de l'arbre est stoppée lorsque la taille de ce dernier dépasse $MinSize$. $Minsize$ est initialisé par la taille de l'arbre minimum crée par ApproximateMinSize sur la même formule. $Minsize$ est modifié lorsque l'on trouve une taille d'arbre inférieure. Notons qu'ApproximateMinSize donne la taille d'un arbre permettant réellement de résoudre la formule $\mathcal{F}$. L'arbre de taille minimum ne peut pas être plus grand que ce dernier.

Malgré son haut degré d'optimisation, la procédure GetMinSize ne peut pas donnée, en un temps raisonnable, la taille de l'arbre minimum pour certaines instances de 75 variables. Afin de remédier à ce problème nous nous restreignons, pour le choix de la variable de branchement, uniquement à l'étude des variables libres apparaissant dans les clauses binaires. En effet, on estime que ces dernières sont en général les meilleures. Cette dernière restriction permet à la procédure GetMinSize de donner la taille minimale de l'arbre de recherche pour des formules de 80 variables.

Nous exécutons la procédure ApproximateMinSize ($k = 1$ et $k \geq 1$ à la racine) ainsi que la procédure GetMinSize pour les instances 3-SAT aléatoires insatisfiables et difficiles. Nous faisons varier n de 50 à 80 (70 pour la procédure GetMinSize Unrestricted) par incrémentation de 5. Pour chaque valeur de n nous résolvons 300 formules. La figure 6 compare les résultats.

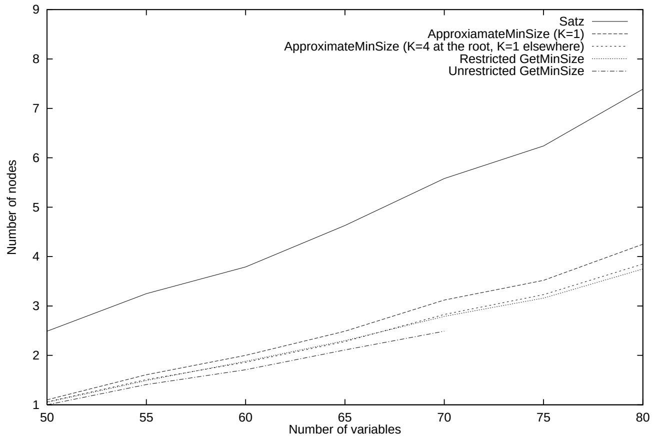

FIG. 6 – *Taille réelle de l'arbre en comparaison avec l'approximation de la taille minimale pour l'arbre de recherche*

Nous ne remarquons pas de différence importante, notamment comparée aux resultats de Satz, entre les résultats de la procédure ApproximateMinSize (surtout pour le cas $k = 4$ à la racine) et ceux de la procédure GetMinSize. Ces derniers résultats confirment la pertinence de la procédure ApproximateMinSize. Notons que si la procédure GetMinSize n'utilisait pas la sous-procédure FurtherSimplification elle aurait pour résultat une taille bien supérieure.

5 Travaux relatifs

La règle de branchement est un facteur clé dans une procédure DP pour la résolution de SAT. Beaucoup de papiers de la littérature se sont focalisés sur des heuristiques utilisées dans les algorithmes DP. Ces papiers proposent soit de nouvelles heuristiques, soit une analyse des heuristiques existantes.

La base commune de plusieurs de ces heuristiques, est l'heuristique MOM's. Elle permet de brancher sur la variable qui possède le maximum d'occurrences dans des clauses de longueur minimale [5, 6, 3, 9]. L'heuristique utilisée dans Satz est une combinaison entre l'heuristique MOM's et des propagations unitaires et est appelée UP heuristique. L'heuristique UP améliore de façon substantielle l'heuristique MOM's. On la retrouve dans C-SAT, cette procédure essaie, grâce à des propagations unitaires expérimentales, de trouver des contradictions aux noeuds du haut de l'arbre. Les premières utilisations de l'heuristique UP sont dues à Freeman dans Posit [6], et à Crawford et Auton dans Tableau [3].

L'analyse d'heuristiques existantes est souvent l'occasion d'en découvrir de nouvelles. Li et Anbulagan ont effectué une étude systématique de l'heuristique UP [12] et en ont proposé une optimale pour Satz. Hooker et Vinay [8] ont analysé de façon probabiliste et expérimentale plusieurs heuristiques et ont proposé une règle de Jeroslow-Wang positive "two-sided". Ils ont également proposé une hypothèse de simplification fournissant des motivations pour une classe d'heuristiques. Bien que la recherche sur les règles de branchement soit plutôt empirique, une étude théorique sur les fondements des heuristiques de branchement est proposée dans [10].

A côté des efforts consentis pour désigner la meilleure règle de branchement, Liberator a montré qu'il est probablement impossible de trouver une règle de branchement capable de sélectionner la meilleure variable possible afin de réduire au maximum l'arbre de recherche. En effet, il a montré que trouver une telle règle était un problème à la fois NP-Difficile et Co-NP-Difficile. Nos résultats suggèrent que même s'il existait une telle règle, elle ne permettrait pas de créer une procédure DP nettement meilleure que celle proposée dans Satz. Ceci dans les cas des instances 3-SAT aléatoires insatisfiables et difficiles.

Beame et al [2] ont établis une borne inférieure, égale à $2^{\Omega(n/\Delta^{4+\varepsilon})}$ avec $\Delta = m/n$, pour la taille de l'arbre de recherche dans le cas des instances 3-SAT aléatoires insatisfiables et difficiles. L'approximation empirique ($2^{n/22.18}$) définie par nos résultats est très loin du résultat théorique ($2^{n/326.2539}$ avec $\Delta = 4.25$). Il est donc encore envisageable d'améliorer la procédure DP en utilisant une méthode substantiellement différente de celle du branchement. Malheureusement, les techniques actuelles de l'IA tels que l'apprentissage, le retour-arrière intelligent ou encore le branchement aléatoire avec une stratégie de redémarrage rapide, bien qu'ayant fait leurs preuves pour les instances SAT structurées [1, 17, 18, 7], ont encore un faible impact pour les instances aléatoires.

Selman et al.[15] ont formulé dix challenges IJCAI en raisonnement et recherche propositionnels. Le premier de ces challenges est de trouver une méthode capable de prouver si une instance 3-SAT aléatoire de 700 variables est insatisfiable. Nos résultats montrent que ce challenge ne pourra probablement pas, à moins qu'une méthode significativement différente du branchement soit trouvée, être résolu par une procédure DP.

Les méthodes autres que DP sont donc encouragées. En fait, les deux autres challenges sur la recherche systématique (par conséquent pour prouver l'insatisfiablité) énnoncés dans [15] consistent à trouver un système de preuves propositionnelles plus puissant que la résolution ,ainsi que de montrer que la programmation linéaire peut être appliquée à SAT. Les réponses à ces deux challenges entraîneront certainement la réponse au premier.

6 Conclusion

Nous utilisons une pseudo procédure DP appelée ApproximateMinSize afin d'approximer, pour les instances 3-SAT aléatoires insatisfiables et difficiles, la taille minimale de l'arbre de recherche d'une procédure DP. Ceci sous l'hypothèse que chaque variable de branchement est la meilleure possible. ApproximateMinSize utilise Satz, la procédure DP la plus rapide, à notre connaissance, sur les instances 3-SAT aléatoires, afin de sélectionner la variable de branchement. Nous fournissons deux arguments pour notre approche.

Premièrement, ApproximateMinSize utilise Satz pour sélectionner plusieurs variables de branchement, ceci pour la racine et ses deux fils. Puis nous comparons les différents arbres obtenus pour ne retenir que celui qui possède la taille minimale. Les résultats expérimentaux montrent une différence négligeable avec le cas où une seule variable est choisie. Ceci signifie que les autres variables choisies ne possèdent probablement pas un intérêt important.

Secondement, nous calculons la taille réelle de l'arbre minimum de recherche d'une procédure DP pour de petites instances 3-SAT aléatoires insatisfiables et difficiles. Nous comparons cette taille avec notre approximation obtenue grâce à ApproximateMinSize. Nous ne remarquons qu'une petite différence entre les deux courbes.

Nos résultat montrent donc que les règles de branchement utilisées actuellement dans le meilleur de l'état de l'art sont déjà très proches de l'optimal pour les instances 3-SAT insatisfiables et difficiles. Ils montrent plus particulièrement que le premier des dix challenges sur la recherche et le raisonnement propositionnels énnoncés lors de l'IJCAI-97 ayant pour nom : prouver qu'une instance 3-SAT insatisfiable et difficile de 700 variables ne pourra probablement pas, à moins qu'une méthode significativement différente du branchement soit trouvée, être résolue par une procédure DP.

Références

[1] Bayardo Jr. R.J., Schrag R.C., *Using CSP Look-Back Techniques to Solve Real-World SAT Instances*,in proceedings of AAAI-97, Providence, Rhode Island, Juillet 1997.

[2] Beame P., Karp R., Pitassi T., Saks M., *On the Complexity of Unsatisfiability Proofs for Random k-CNF Formulas*, STOC98.

[3] Crawford J.M., Auton L.D., *Experimental Results on the Crossover Point in Random 3-SAT*,Artificial Intelligence, no. 81, 1996.

[4] Davis M., Logemann G., Loveland D., *A Machine Program for Theorem Proving*, In Commun. ACM 5, 1962, pp. 394-397.

[5] Dubois O., Andre P., Boufkhad Y., Carlier J., *SAT Versus UNSAT*. Second DIMACS Challenge: Cliques, Coloring and Satisfiability. Rutgers University, NJ, 1993.

[6] Freeman J.W., *Improvements to Propositional Satisfiability Search Algorithms*, Ph.D. Thesis, University of Pennsylvania, 1995.

[7] Gomes C.P., Selman B., Kautz H. *Boosting Combinatorial Search Through Randomization*, In proceedings of AAAI'98, 1998.

[8] Hooker J.N., Vinay V. *Branching rules for satisfiability*, Journal of Automated Reasoning, 15:359-383, 1995.

[9] Jeroslow R., Wang J. *Solving propositional satisfiability problems*, Annals of Mathematics and AI 1(1990), pp. 167-187.

[10] Kullmann O, *Heuristics for SAT algorithms: Searching for some foundations*, submitted to Discrete Applied Mathematics, 23 Pages.

[11] Li C.M., *A constraint-based approach to narrow search trees for satisfiability*, Information Processing Letters 71 (1999) 75-80.

[12] Li C.M., Anbulagan, *Heuristics Based on Unit Propagation for Satisfiability Problems*, In Proceedings of IJCAI-97, Page 366-371, Nagoya, Japon, Août 1997.

[13] Liberatore P., *On the complexity of choosing the branching literal in DPLL*, Artificial Intelligence, Vol 116, Issue 1-2, Pages 315-326, 2000.

[14] Mitchell D., Selman B., Levesque H., *Hard and Easy Distributions of SAT Problems*, In Proceedings of AAAI-92, San Jose, CA, July 1992, pp. 459-465.

[15] B. Selman, H. Kautz, D. McAllester *Ten Challenges in Propositional Reasoning and Search*, in proceedings of IJCAI-97, Nagoya, Aichi, Japan, August 1997.

[16] Selman, B., Kautz H., Cohen B., *Noise strategies for local search*, In Proc. of AAAI-94, pp. 337-343. AAAI Press/The MIT Press, 1994.

[17] Silva J. P. M., Sakallah K. A., *Conflict Analysis in Search Algorithms for Propositional Satisfiability*, Proceedings of the International Conference on Tools with Artificial Intelligence, Novembre 1996.

[18] Zhang H., *An efficient propositional prover*, In Proc. of int' Conf. on Automated Deduction, pp272-275, Juillet 1997.