Davis et Putnam, 40 ans après : une première expérimentation

Philippe Chatalic, Laurent Simon*

1^er Mars 1999

L.R.I., U.A. CNRS 410,
Université Orsay Paris XI

Résumé

Dans cet article, nous présentons une implémentation de la procédure de Davis et Putnam telle qu'elle fut présentée en 1960, sur laquelle nous menons une étude expérimentale. À notre connaissance en effet, un consensus sur son innefficacité s'est installé dans la littérature sans pour autant avoir testé de manière pratique quelles étaient ses réelles limites. En menant un telle étude, nous montrons que même si elle répond à un problème bien plus complexe que la procédure du même nom avec retour arrière, elle offre des résultats surprenants sur certains exemples de DIMACS, en dépassant les performances de beaucoup d'algorithmes spécialisés pour SAT. Enfin, cet article permet de caractériser certains problèmes propres de cette approche par une étude approfondie de son comportement sur des instances aléatoires.

1 Introduction

En 1960, Davis et Putnam [6] proposent un algorithme permettant de prouver l'insatisfiabilité d'un formule propositionnelle. Deux ans plus tard, Davis, Logemann et Loveland [12] en présentent une version modifiée qui, aujourd'hui encore, reste la procédure la plus utilisée pour la résolution du problème SAT. Pourtant, ces dix dernières années ont vu un nombre important de travaux traitant du problème SAT, tant sur le plan théorique [15] qu'expérimental [5, 1]. Stimulés par les progrès matériels, ces travaux ont permis des améliorations constantes dans la taille des problèmes traités, voire significatives à l'aide de méthodes incomplètes. Mais, il est frappant de constater que la majorité des algorithmes complets est toujours basés sur la procédure de 1962 [12]. Les deux versions ([6, 12]) sont cependant très différentes dans leurs principes. Bien que s'appuyant toutes deux sur la résolution, celle de 1960 consiste en une suite de saturations dans le but d'éliminer une à une les variables, alors que celle de 1962 correspond à une recherche avec retour arrière couplée à un mécanisme de propagation unitaire. Cette dernière étant bien moins coûteuse en mémoire, la puissance des machines de l'époque explique probablement la rapide mise à l'écart de la version initiale. Cet oubli a probablement été accentué par le fait que de nombreux auteurs confondent dans la littérature les deux versions et appellent «procédure de Davis et Putnam» la version de Davis, Logemann et Loveland. Pour éviter cette confusion, nous référencerons dans la suite la première version par DP et la seconde par DPL.

S'il est vrai que pour résoudre le problème SAT, la version initiale paraît de prime abord plus coûteuse, elle n'en est pas pour autant dénuée d'intérêt. Par exemple, les informations qu'elle gère peuvent permettre de reconstruire l'ensemble de *tous* les modèles d'une formule et ce, dans un temps proportionnel à leur nombre. On peut aussi limiter le nombre de variables éliminées par DP pour produire des contraintes nécessairement satisfaites par les variables restantes, ce qui peut s'avérer intéressant dans un contexte de validation ou de compilation de bases de connaissances [14].

À notre connaissance cependant, il semble que personne ne se soit intéressé à cette procédure, mis à part Dechter et Rish [8] ainsi que les travaux, menés indépendamment, d'Édf [10], qui avaient remarqué l'interêt de DP dans un contexte de validation et dans la continuité desquels ce situe notre travail.

L'objet de cet article est de rendre compte d'une telle expérimentation. Après une présentation du principe de l'algorithme de DP et de ce qui le différencie de DPL dans la section 2, nous présentons les principaux

*(chatalic, simon)@lri.fr

choix liés à la mise en œuvre de DP. Dans la section 4, nous présentons les résultats des expérimentations que nous avons conduites, d'une part sur des exemples aléatoires et d'autre part sur des problèmes de référence issus de la littérature sur SAT. Les premiers permettent d'appréhender les limites en taille d'une telle approche pour SAT et font apparaître des résultats contre-intuitifs dans le cadre d'une élimination partielle. Les seconds sont beaucoup plus surprenant dans la mesure où, sur plusieurs classes d'exemples, ils surpassent les résultats obtenus par certains des meilleurs démonstrateurs basés sur DPL sans mémoire des conflits, alors que DP semble résoudre un problème bien plus complexe que DPL.

2 Algorithmes basés sur DP

Dans cette section sont présentés les algorithmes de Davis et Putnam (DP) de 1960[6], son implémentation proposée par Dechter et Rish en 1994 [8] et la légère variante *Calcres* [14] que nous proposons. La procédure de Davis, Logeman et Loveland publiée en 1962 [12] ainsi que les liens entre les deux versions sont ensuite abordés.

Préliminaires : Les *variables propositionnelles* sont notées par les lettres majuscules P, Q, ... , les littéraux ($\neg P$, P) par des lettres minuscules (p, q, ...). Les *clauses* (conjonctions de littéraux) sont notées par les lettres grecques α, β, ... On appelle *clause unitaire* une clause ne contenant qu'un littéral. La *résolution* est l'opération qui à deux clauses $(\alpha \vee Q)$ et $(\beta \vee \neg Q)$ associe la clause $(\alpha \vee \beta)$ ne contenant plus la variable Q et conséquence logique des deux premières. On appelle *coupure* sur Q l'opération consistant à effectuer toutes les résolutions possibles sur Q. Une formule φ est sous forme normale conjonctive (CNF) si elle s'écrit comme une conjonction de clauses. Un *modèle* est une assignation des variables apparaissant dans la formule à $\{vrai, faux\}$ dont l'évaluation sur la formule est égale à $vrai$.

Davis et Putnam (DP) : DP, tel qu'il fut présenté en 1960, peut s'écrire comme décrit figure 1. Son principe est assez simple. À chaque itération, la variable P choisie est éliminée de la nouvelle formule par coupure.

Entrée : une formule CNF φ construite sur un ensemble de variables Ω.
Sortie : clause vide (φ insatisfiable) ou formule vide de clauses (φ satisfiable)
DP (φ, Ω)
  si $\Omega = \emptyset$ *retourner* φ
  choisir une variable P *dans* Ω
  retourner **DP** ($(\varphi \wedge \neg P) \vee (\varphi \wedge P)$,
    $\Omega \setminus \{P\}$)

FIG. 1: Algorithme de Davis et Putnam originel (DP)

Calcres : Dans [14, 10] est présenté une variante de DP dans laquelle toutes les variables de la formule initiale ne sont pas éliminées. Cette variante est utilisée pour expliciter des contraintes, présentes implicitement dans une formule φ initiale, sur un sous-ensemble Ψ des variables. La formule résultant de cette élimination est appelée *résolvant* de φ sur Ψ. Notons qu'aucune supposition sur la primalité du résolvant de φ n'est faite. Notre implémentation est basée sur l'opération de *distribution* entre deux ensembles de clauses, correspondant à la mise sous CNF d'une disjonction de deux formules sous CNF. DP peut ainsi se réécrire comme Calcres, décrit figure 2, appliqué à l'ensemble des variables d'une formule.

Dans [10] est établi le théorème suivant qui caractérise l'importance du calcul de résolvant dans la compilation de bases de connaissances :

Théorème 1 (Prolongement) *Soit φ sous cnf, construite sur les variables Ω et φ_Ψ le résolvant de φ sur $\Psi \subseteq \Omega$, alors pour pour tout modèle de φ_Ψ, il existe au moins un modèle de φ dont l'affectation des variables dans Ψ est identique à celle du modèle de φ_Ψ. On dit que tout modèle de φ_Ψ peut être prolongé par au moins une affectation des variables de $\Omega \setminus \Psi$ à un modèle de φ.*

Directional resolution (DR) : Dans [8], les auteurs proposent une implémentation de DP pour la résolution de SAT. Celle-ci est présentée comme une compilation de la formule initiale φ en une théorie $E_d(\varphi)$ dans un ordre prédéfini d'élimination des variables, comme donné figure 3. L'ordre d'élimination des variables

Entrée : une formule CNF φ construite sur un ensemble de variables Ω,
un sous ensemble Ψ des variables de Ω.
Sortie : le résolvant de ϕ sur Ψ.
Calcres (φ, Ψ)
choisir une variable P *dans* Ψ
soit φ_P^+ (resp. φ_P^-) la conjonction des clauses de φ contenant
le littéral $p = P$ (resp. $p = \neg P$) dont l'occurrence de P a été enlevée,
soit $\varphi_{P'}$ la conjonction des clauses de φ ne contenant pas P,
retourner **Calcres** ($(\varphi_P^+ \vee \varphi_P^-) \wedge \varphi_{P'}$, $\Psi \setminus \{P\}$)

FIG. 2: Algorithme de calcul de résolvant de φ sur Ψ, (Calcres)

est fixé et déduit de l'analyse de la formule initiale, même si les auteurs précisent qu'un ordre dynamique reste possible.

L'apport de Dechter et Rish est multiple. D'abord, ils montrent que les classes de problèmes *2-cnf* et *horn* sont polynomiaux pour DR. Ils construisent ensuite une classe de problème difficile pour leur implémentation de DPL et facile pour DP, les *k-chains*, même si on verra dans la partie 4.1 que nos propres résultats sont beaucoup moins probants. De plus, ils montrent que DR compile une théorie permettant d'exhiber n'importe quel modèle dans un temps $O(|E_d(\varphi)|)$, ce qui signifie *tous les modèles en temps proportionel au produit de leur nombre par* $|E_d(\varphi)|$ — dans Calcres, il suffirait de garder l'ensemble des $(\varphi_{P_i}^+, \varphi_{P_i}^-)$ pour obtenir un tel résultat —. Enfin, ils montrent que le temps de calcul de DR est de l'ordre de $O(n.|E_d(\varphi)|^2)$. Même si leur théorème se révèle faux lorsque φ est insatisfiable, et semble une borne très supérieure à la réelle complexité de DR, il montre, dans le cas φ satifiable un certain lien entre le temps de calcul de DR et la taille de la théorie, une fois compilée (ce qui n'est pas sans poser un certain problème quant à la gestion des subsomptions dans $E_d(\varphi)$).

Entrée : une formule CNF φ construite sur un ensemble de variables Ω.
un ordre $d = Q_1, \ldots, Q_n$ sur Ω
Sortie : une théorie $E_d(\varphi)$
DR (φ, Ω, d)
1. soit la partition $c_1, \ldots, c_n$ sur les clauses, c_i contenant toutes les clauses dont le plus haut littéral dans l'ordre d est Q_i.
2. *Pour* $i = n$ *à* 1 *faire* :
3. Résoudre chaque paire $\{(\alpha \vee Q_i), (\beta \vee \neg Q_i)\} \subseteq c_i$
4. *Si* $\gamma = \alpha \vee \beta$ est vide, *retourner* $E_d(\varphi) = \emptyset$
5. *Déterminer l'index* j *de* γ et l'ajouter à c_j.
6. *Fin-pour*
7. *Retourner* $E_d(\varphi) = \bigcup_i c_i$.

FIG. 3: Algorithme Directional Resolution (DR)

2.1 Algorithme de Davis, Logemann et Loveland

Dans [12], les auteurs présentent DPL — qui n'est rien d'autre que DP dans lequel on a transformé le connecteur logique $\vee$ en sa valeur sémantique : *ou* —, que l'on peut écrire comme figure 4.

Entrée : une formule CNF φ construite sur un ensemble de variables Ω.
Sortie : *SATISFIABLE* ou *INSATISFIABLE.*
DPL (φ, Ω)
faire toutes les propagations unitaires dans φ.
si φ contient la clause vide, *retourner INSATISFIABLE*
si $\Omega = \emptyset$, *retourner SATISFIABLE*
choisir un littéral p *d'une variable* P *dans* Ω
si **DPL** ($(\varphi \wedge p)$, $\Omega \setminus \{P\}$)= *SATISFIABLE*,
retourner SATISFIABLE
sinon retourner **DPL** ($(\varphi \wedge \neg p)$, $\Omega \setminus \{P\}$)

FIG. 4: Agorithme de Davis, Logeman et Loveland (DPL)

3 Implémentation de Calcres

À notre connaissance, la seule implémentation de DP qui ait été proposée jusqu'ici est donc *Directional Resolution* de Dechter et Rish. Même si leur but n'a pas été d'optimiser leur implémentation, il paraît difficile de situer aujourd'hui un algorithme dans la littérature SAT sans utiliser des structures de données adaptées. C'est pourquoi nous décrivons ici brièvement les grandes lignes de notre implémentation, qui a été développée en *Objective Caml*, un langage de la famille ML permettant d'obtenir une compilation en *code natif* des programmes.

Heuristiques : Le choix d'une variable à couper se fait à chaque itération, de manière dynamique, en préférant celle qui est supposée entrainer la moins grande difficulté dans la suite du calcul. Notre implémentation peut prendre en compte différentes approches : le nombre de clauses produites, la longueur moyenne ou maximale de ces clauses ou encore la somme de leurs longueurs.

L'importance de la présence d'une heuristique appropriée s'est rapidement fait sentir, dans la mesure où sans elle, pratiquement aucun des exemples traités dans la suite de cet article ne peut aboutir en raison du nombre trop important de clauses manipulées. Par contre, les différences entre elles ne sont pas claires et malgré nos efforts dans le but de multiplier les approches heuristiques, rares sont les exemples permettant de les discerner (aucun dans les exemples aléatoires uniformes). Dans la suite, sauf mention contraire, nous utiliserons l'heuristique choisissant la variables dont la coupure entraîne une estimation du surplus de clauses produites minimal.

Notons que dans ce cas-là, si f_P^+ est la fréquence[1] positive de la variable P et f_P^- sa fréquence négative, en estimant le nombre de clauses produites par coupure sur P comme égale à

$$H(P) = f_P^+ * f_P^- - f_P^+ - f_P^-,$$

la variable qui, à chaque itération, semble la plus *intéressante* pour DP, semble la plus *inintéressante* pour DPL [9]. C'est en effet celle qui, après affectation positive ou négative entraîne le moins de simplifications dans la formule. On retrouve cependant un cas particulier commun : les deux algorithmes utilisent la propagation unitaire (dans DP, si une variable P pure apparaît sous une seule forme de littéral, $H(P)$ est alors négatif : cette variable est choisie de manière prioritaire par rapport aux variables non pures).

Utilisation de tries : À chaque coupure des tests de subsomptions sont à effectuer sur les clauses produites. On peut tenter de réduire le coût de ces tests en utilisant une structure de donnée adaptée à la manipulation d'ensembles de clauses comme on peut en trouver dans la littérature traitant d'impliqués premiers [7] ou dans certains algorithmes DPL réçents [16]. Dans notre implémentation, nous utilisons donc des *tries*, une représentation arborescente d'un ensemble de clause facilitant les tests et l'élimation des subsomptions lors de l'ajout de nouvelles clauses, en implémentant l'ordre sur les variables du *trie* proposé dans [7].

4 Expérimentation de Calcres et DP

Nous présentons dans cette partie les résultats de l'étude menée sur DP au travers d'une étude de Calcres. Notre démarche, répondant aux critères expérimentaux proposés dans [11], suit plusieurs étapes. D'abord, nous comparons notre implémentation avec DR, la seule implémentation de DP qui, à notre connaissance, ait été proposée. Pour cela, nous reprenons tous les exemples testés dans [8]. Ensuite, nous tentons de nous positionner par rapport à DPL, algorithme qui a l'avantage d'avoir été étudié fortement, même si, DP répondant à un problème plus difficile, ce dernier devrait, *a priori*, présenter des performances bien inférieures à DPL sur SAT. Nous utilisons pour cela une approche similaire à une telle étude [5, 3] en analysant le comportement de DP sur des instances aléatoires uniformes et sur nombre de classes d'exemples issus de la base de tests DIMACS. La particularité du comportement de DPL sur les formules aléatoires uniformes, en présentant un phénomène net de seuil, nous pousse dans une première partie à observer si, d'une part DP présente la même caractéristique et, d'autre part s'il ne montre pas de difficultés propres. L'étude sur ces formules permet en même temps d'appréhender une borne dans la taille des problèmes aléatoires uniformes traitables. Nous cherchons aussi, pour ces applications éventuelles dont nous avons parlé dans l'introduction, à étudier Calcres sans éliminer toutes les variables, sur des exemples aléatoires uniformes et présentons des résultats quelque peu contre-intuitifs concernant le calcul de résolvant sur de telles instances.

[1] On appelle fréquence positive (resp. négative) d'une variable le nombre d'occurrence de cette variable sous forme positive (resp. négative) dans la formule

Notre étude se présente en trois parties. La première compare les implémentations de DP entre DR et Calcres. La seconde traite des expérimentations sur des exemples aléatoires uniformes à la fois sur DP (Calcres avec toutes variables éliminées) et sur Calcres (élimination d'un sous-ensemble des variables). La dernière partie rapporte le comportement de Calcres sur des classes de DIMACS.

Algorithmes DPL utilisés : Les deux implémentations de type DPL que nous avons utilisées sont *Asat* d'Olivier Dubois [9] et *sato3.0* de Hantao Zhang[3]. Ces deux algorithmes ont été developpé en C. La particularité de sato3.0 est d'utiliser un backjumping permettant de mémoriser d'anciens conflits pour éviter des affectations vouées à l'échec. Ce dernier est, à notre connaissance, le meilleur démonstrateur SAT complet en ce qui concerne les problèmes ayant une structure (par exemple sur la base de tests DIMACS). Asat, certainement pour sa simplicité et sa rapidité, surpasse cependant sato3.0 sur les exemples aléatoires. L'utilisation de ces deux algorithmes donne une bonne idée de l'état de l'art concernant les démonstrateurs SAT issus de DPL.

4.1 Calcres et DR : différence de temps exponentielle

La figure 5Fig. 5(a): , d'échelle logarithmique, montre une différence de temps exponentielle entre l'implémentation de Dechter et Rish et celle que nous présentons sur *3-cnf*. Même si un écart de temps linéaire est dû aux performances des machines, de 5 ans plus récentes, le caractère exponentiel n'est pas remis en cause. Les raisons d'une telle différence sont certainement l'utilisation de *tries* et d'heuristiques de choix dynamique.

Sur les k-chains : Dans leur article, Dechter et Rish démontrent, à la fois sur le plan théorique et pratique, le bon comportement de DP sur une classe de problèmes appelé *k-chains*. Nous avons implémenté un générateur de ce type de problème. Mais, à la différence des résultats de Dechter et Rish, nous n'avons pas observé un comportement de DPL aussi mauvais que dans [8]. Même si Asat ne prend pas en compte la structure du problème comme sato3.0, Asat est resté constamment beaucoup plus performant que Calcres, qui lui-même était plus performant que DR. Encore une fois, nous pensons que les raisons de telles différences entre Calcres et DR d'une part et de l'implémentation DPL de Dechter et Rish et Asat d'autre part viennent essentiellement des implémentations et des heuristiques.

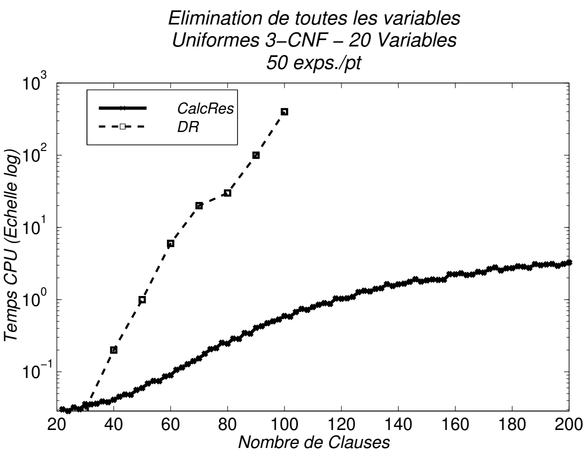

Fig. 5(a): Comparaison Calcres/DR et évolution du temps CPU(s) de Calcres selon *nbc*

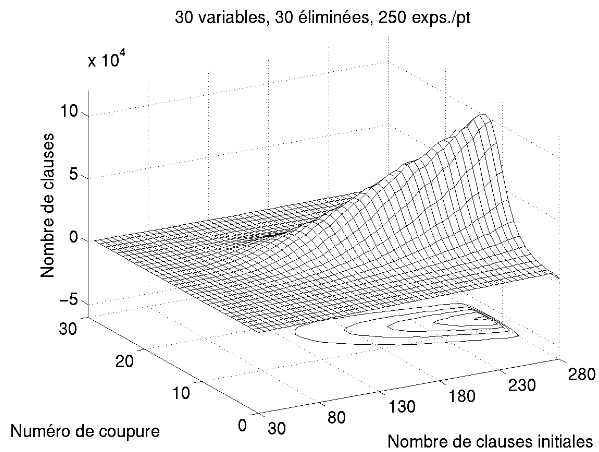

Fig. 5(b): Évolution du nombre de clauses manipulées pendant chaque calcul selon *nbc*

4.2 Bases de tests aléatoires uniformes

Comme utilisé dans [5], notre générateur d'instances aléatoires uniformes de clauses de longueurs constantes consiste, selon un nombre de clauses et un nombre de variables données, à tirer chaque variable avec la même chance, en la niant avec la probabilité 1/2. Les seules particularités de notre générateur sont d'autoriser des longueurs de clauses variables (selon un maximum et un minimum donnés), d'empêcher l'apparition en double des variables dans chaque clause (*i.e.* tautologies et clauses raccourçies) et l'obligation pour chaque variable d'apparaître au moins une fois dans la formule. On appelle *k-cnf* les formules générées de telle sorte et dont toutes les clauses sont de longueur k.

a. Le phénomène de seuil observé sur *3-cnf* **:** Ce comportement a été très étudié dans la littérature. Dans [5], les auteurs montrent expérimentalement que ce seuil est situé autour d'un quotient *nombre de clauses* sur *nombre de variables* de 4.28. En dessous de 3.5, pratiquement toutes les instances sont satisfiables, et au-delà de 5.0, presque toutes insatisfiables. Le temps de calcul des algorithmes DPL semblent lié de manière très étroite à ce *ratio*, les instances *difficiles* étant situées près du seuil.

Une première chose à observer sur DP dans le but de le comparer à DPL est donc le lien entre ce *ratio* et le temps de calcul de notre implémentation. La figure 5Fig. 5(a): schématise une telle évolution, en montrant très clairement, avec un *ratio* de 1 à 10 que *plus il y a de contraintes, plus un exemple 3-cnf est difficile pour DP.*

Pour tenter de comprendre une des raisons de ce comportement pour DP nous représentons sur la figure 5Fig. 5(b): l'évolution du nombre de clauses manipulées (hauteur), à chaque itération de DP (profondeur), selon le nombre de clauses initiales (largeur) pour des formules initiales de 30 variables.

Analyse : Il est clair que DP est insensible au seuil qui pourtant se retrouve dans les autres méthodes, complètes ou incomplètes[2]. La figure 5Fig. 5(b): montre bien l'accroissement régulier du nombre de clauses manipulées selon le nombre de clauses initiales, lié à l'accroissement de la difficulté. On peut expliquer ce comportement par le fait que, *à aucun moment, DP ne cherche à prouver l'inconsistance de la formule initiale.* De plus, en rendant explicites des contraintes implicites, par l'ajout de nouvelles clauses déduites, ces dernières sont d'autant plus nombreuses que les contraintes initiales le sont. Même si on peut penser que l'ajout de contraintes dans la formule initiale permet de déduire plus rapidement la clause vide — ce qui explique que la crête observée sur la figure 5Fig. 5(b): est atteinte de plus en plus tôt dans l'ordre des coupures (la moyenne des 250 expérimentations prend en compte les formules composés d'une seule clause (clause vide) trouvées de plus en plus tôt) —, on peut dire que, *pour DP, la difficulté induite par l'ajout de contraintes augmente bien plus rapidement que la facilité avec laquelle on déduit la clause vide.*

b. Influence de la taille de ψ pour Calcres : ψ étant l'ensemble des variables sur lesquelles nous calculons le résolvant, nous pouvons nous attendre à ce que plus ψ soit grand (donc moins on a de variables à éliminer par coupure), plus le calcul de résolvant est facile. La figure 5Fig. 5(e): montre qu'il n'en est rien.

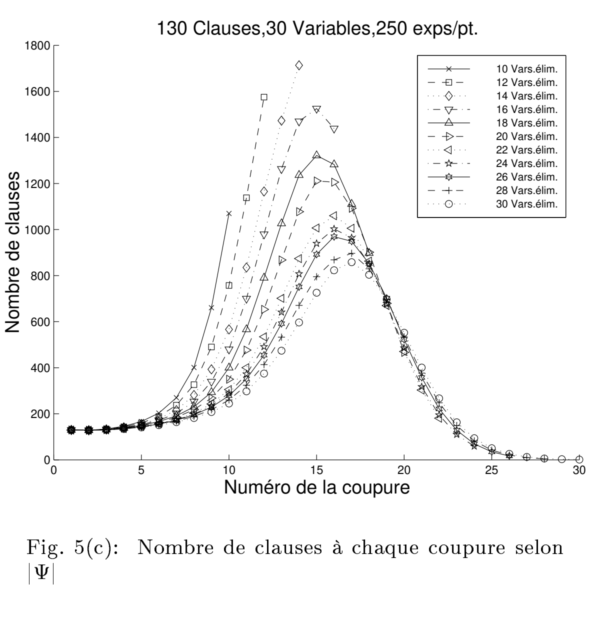

Fig. 5(c): Nombre de clauses à chaque coupure selon $|\Psi|$

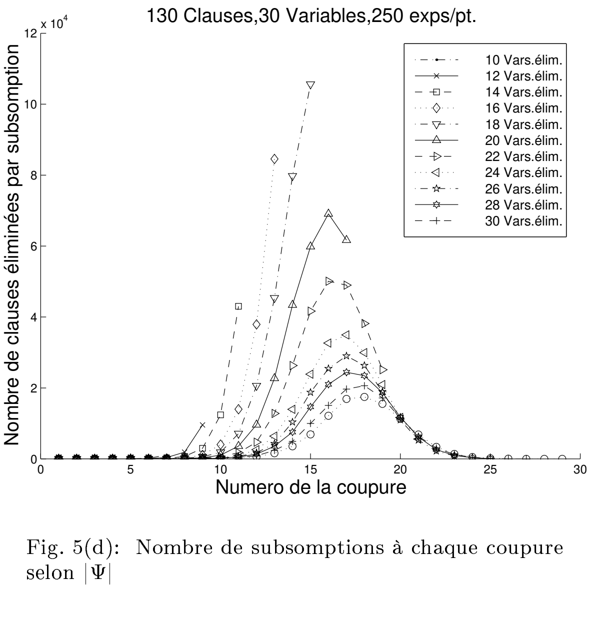

Fig. 5(d): Nombre de subsomptions à chaque coupure selon $|\Psi|$

Analyse : D'après nous, *la plus grande difficulté dans le calcul de résolvant d'instances aléatoires, qui se situe autour de la moitié des variables à éliminer, peut être inculqué à la liberté laissée aux heuristiques.* La figure 5Fig. 5(c): montre bien en effet que cette difficulté est dûe à un nombre de clauses augmentant beaucoup plus rapidement dans un cas que dans l'autre. Or la différence entre les deux vient uniquement du choix laissé à l'heuristique.

La seconde caractéristique de cette difficulté est la grande différence dans le nombre de clauses produites puis éliminées par subsomption selon le nombre de variables initiales à éliminer, comme illustré figure 5Fig. 5(d): : à 14 variables à éliminer, la 14ième coupure entraîne plus de 110000 clauses éliminées par subsomption, et seulement 250 gardées, alors qu'à 30 variables à éliminer, la 14ième coupure entraîne 200 clauses subsumées et 100 gardées. En quelque sorte, on peut s'estimer heureux, grâce au nombre énorme d'élimination par subsomption, de n'avoir qu'*un si petite accroissement du nombre de clause pour* $|\Psi| = |\Omega|/2$.

Une explication peut venir de la distribution de clauses par longueur nettement différente dans un cas par rapport à l'autre, comme on le voit sur le tableau 1. Il est remarquable que, malgré le nombre important de clauses de longueur 6 dans le cas $|\Psi| = |\Omega|/2$, aucune clause de longueur 10 (correspondant à une résolution sur deux clauses de longueur 6) n'est ajouté à la formule, donc toutes sont éliminées par subsomption. S'il en est de même dans le cas $|\Psi| = |\Omega|$, *la différence importante entre le nombre de clauses longues explique, au moins en partie, la différence dans le nombre de subsomptions.* De plus, de manière plus intuitive, on pourrait penser que cette différence vienne du fait que beaucoup de clauses ont été produites sur peu de clauses initiales, en peu de coupure. Ceci pourrait entraîner un *mélange* des littéraux dans les clauses non homogène, donc conduisant à beaucoup de répétitions entraînant aussi ce nombre important de subsomptions.

Numéro de coupure	Nombre de variables à éliminer	Nombre de clauses de longueur									
		1	2	3	4	5	6	7	8	9	10+
14	14	0	20	50	190	490	600	350	100	10	1
14	30	0	20	50	120	200	200	50	20	0	0

TAB. 1: Moyenne des types de formules manipulées selon $|\Psi|$

c. Type de résolvants obtenus par Calcres selon $|\Psi|$: Lorsque l'on veut obtenir un résolvant sur un certain sous-ensemble de variables, on peut se demander si, par exemple, celui-ci ne peut pas être obtenu par succession de tirages au sort de clauses construites sur les variables restantes, suivi du test de l'inconsistance de la conjonction de la formule initiale avec la négation de la clause tirée au sort. Si la formule est inconsistante, la clause tirée est logiquement impliquée par la formule initiale, donc par son résolvant. [13] utilisent une telle technique pour trouver des impliqués de longueur 2. La figure 5Fig. 5(f): donne l'idée du type de résolvant selon le nombre de variables à éliminer en représentant les différentes longueurs de clauses présentes.

Analyse : On voit qu'*une telle technique de génération de résolvant sur des instances aléatoires semble impossible* tant que le nombre de clauses dans le résolvant est important, c'est à dire au moins sur des résolvants construits sur 6 à 28 variables sur les 30 initiales, d'autant plus qu'il s'agit de clauses longues (à 14 variables à éliminer, on a 600 clauses de longueur 6 sur plus de 230 millions possibles).

4.3 Les Benchmarks de DIMACS

La base de tests DIMACS répertorie des instances de problèmes difficiles pour SAT. À la différence des instances *k-cnf*, celles-ci ont chacune une certaine structure. Même si l'expérimentation de démonstrateurs SAT sur des formules aléatoires a permis une amélioration significative de leurs possibilités, les modèles de génération uniforme des instances aléatoires posent un sérieux problème. Rien ne prouve en effet que l'échelle de comparaison entre démonstrateurs construite sur les instances aléatoires restera valide sur les instances ayant une certaine structure. C'est pourquoi il est tout aussi important, sinon plus, de recueillir des bases d'exemples difficiles non aléatoires-uniformes pour les démonstrateurs SAT, en attendant que le dixième challenge posé dans [4], *i.e. To develop a generator for problem instances that have computational properties that are more similar to real-world instances*, soit résolu.

a- Temps CPU sur certains problèmes : Le tableau 2 rend compte de la somme des temps CPU nécessaires pour résoudre des problèmes de DIMACS, sur un *Pentium II-400* sous Linux avec 256Mo de mémoire. Pour chaque problème sont donnés son nom et son nombre d'instances. Pour chaque démonstrateur sont donnés la somme des temps de traitement (en assumant qu'un exemple non résolu en moins 10000s compte pour 10000s) ainsi que le nombre d'instances résolues. Nous ne faisons apparaître ici que les problèmes pour lesquels Calcres

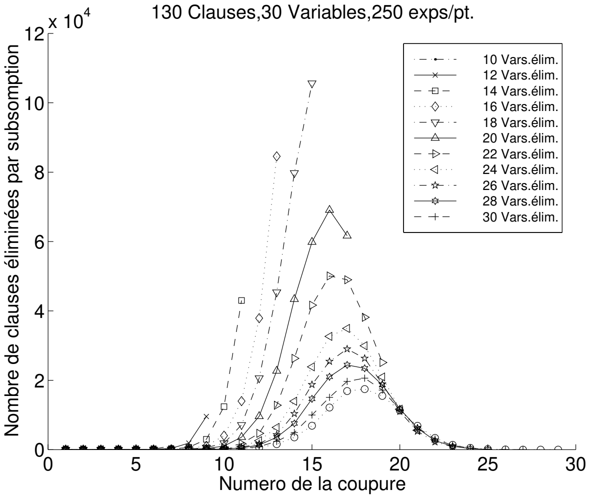

Fig. 5(e): Temps de Calcres selon $|\Psi|$

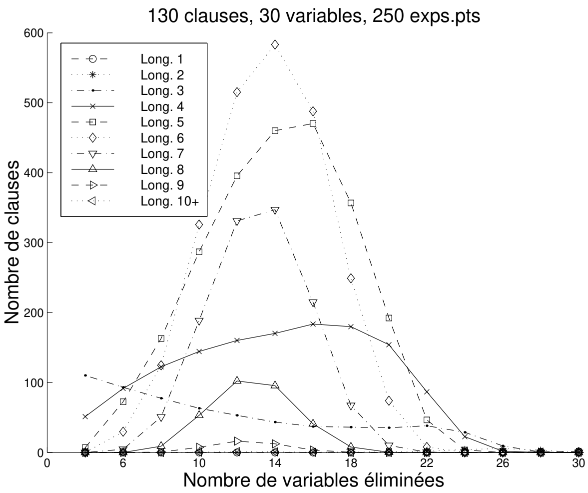

Fig. 5(f): Types de résolvant, par longueur de clause, selon $|psi|$

a été en mesure de traiter au moins une instance, ce qui risque d'en biaiser une lecture superficielle en faveur de Calcres. On ne trouve pas, par exemple, les problèmes *parity16*, *parity32*, *flat-50* et *aim-150* dont beaucoup d'instances sont pourtant résolues par Asat ou sato3.0 rapidement.

Problèmes	Asat	sato3	Calcres	Problèmes	Asat	sato3	Calcres
Dubois 13	52770 9	0,12 13	1,25 13	flat30 100	0,35 100	0,34 100	119,1 100
pret60 4	110 4	0,11 4	0,75 4	aim-50 24	0,58 24	0,14 24	80940 16
ssa 8	20323 6	1,42 8	3767 8	aim-100-1,6 8	3693,3 8	0,04 8	7299 8
ii8 14	47892 10	0,31 14	13000,08 1	par8 10	0,13 10	0,06 10	14,1 10
hole 5	155 5	163 5	40215 1	bf 4	30101 1	1,18 4	20424 2

TAB. 2: Tableau des temps CPU sur des instances de DIMACS

Analyse : Notre propos n'est pas ici de démontrer que DP est meilleur que DPL pour SAT, ce qui semble totalement erroné lorsque DPL est aidé d'un backjumping efficace comme dans sato3.0. Cependant, les résultats obtenus sont bien supérieurs à ceux auxquels on pouvait s'attendre, dans la mesure où *DP est plus performant que Asat sur nombre d'instances.* Pour pouvoir comparer Calcres et sato3.0, rappelons tout de même que sato3.0 a été présenté en 1997 après des années de travail dans une communauté SAT effervescente. Jusqu'alors, et sur certaines classes de problèmes, DP aurait été mieux classé que la plupart des algorithmes DP présents dans la littérature. Ce n'est certes plus le cas, mais DP répondant à un problème bien plus difficile que DPL, la différence de temps avec sato3.0 sur certaines instances semble compensée par l'écart dans la complexité des problèmes auxquels chacun des deux répondent.

Si l'on veut poursuivre la comparaison, une question que l'on peut se poser est de savoir si les problèmes de DIMACS n'ont pas seulement pour but de mettre en échec la technique de retour arrière présent dans DPL (comme semblent l'être les *k-chains*) ? Si c'était le cas, résoudre ces problèmes par une autre méthode n'apporterait pas forcément quelque chose de satisfaisant (on ne compare pas les performances de deux coureurs si l'un d'eux utilise autre chose que ses jambes !). Au vu des multiples problèmes différents sur lesquels Calcres est plus efficace que Asat, cela paraît peu probable. De plus, certains problèmes sont issus de problèmes concrets. Nous pensons donc que, *sur certains exemples ayant une certaine structure, la prise en compte de la structure* *globale du problème à chaque itération, comme le fait Calcres, est plus efficace qu'une série de prises en compte partielle, comme dans DPL sans mémoire des conflits.*

b- Comportement de DP sur certaines instances :

Comme nous l'avons fait pour *3-cnf*, nous illustrons quelques comportements de Calcres sur des exemples de DIMACS. On peut observer, sur les figures **??**, 5Fig. 5(g): et 5Fig. 5(h): l'évolution du nombre de clauses pendant les calculs.

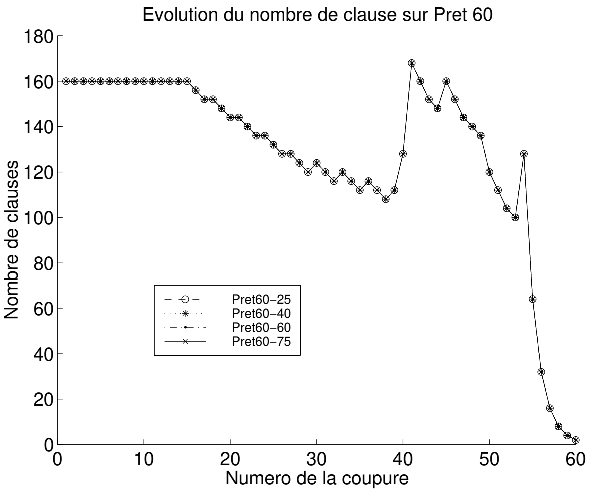

Fig. 5(g): Évolution du nombre de clauses sur *pret60*

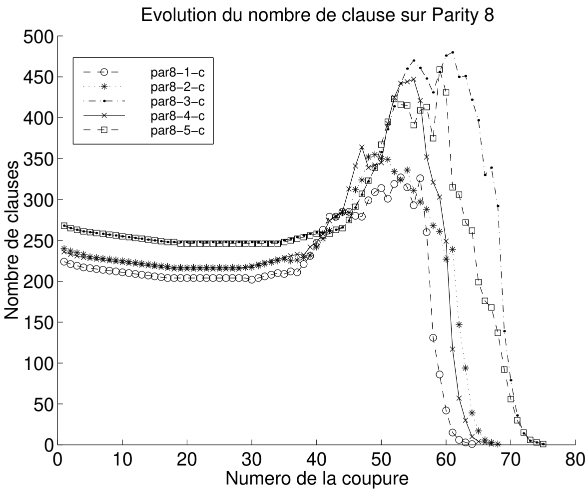

Fig. 5(h): Évolution du nombre de clauses sur *par8*

Analyse : La première observation est la différence flagrante dans le comportement de Calcres sur des instances aléatoires uniformes et sur les instances de DIMACS. Ceci peut être expliqué par le fait qu'à tout moment, *c'est la structure globale du problème qui est manipulée par Calcres*. Or, il est clair que la structure d'un problème aléatoire uniforme et celle d'un exemple de DIMACS diffèrent largement.

Il paraît ensuite surprenant que toutes les instances de *pret60*, figure 5Fig. 5(g): , conduisent exactement à la même évolution du nombre de clauses. Dans ce cas précis (unique à notre connaissance), *Calcres semble caractériser exactement toutes les instances du même problème* pret, a posteriori, *par son comportement lors du calcul de résolvant*. Sans aller dans cette extrême où les évolutions du nombre de clauses sont exactement identiques, il est vrai que les courbes obtenues lors d'un calcul sont souvent assez caractéristiques pour retrouver la classe de problème à laquelle appartient l'instance traitée par l'observation de cette seule courbe.

Enfin, rappelons que ces évolutions ne doivent pas cacher le fait que certaines formules voient leur nombre de clauses diminuer en même temps que la longueur moyenne des clauses augmente. Une diminution du nombre de clause ne signifie pas forcément une diminution de la *complexité* de la formule, ce qui se caractérise souvent (figures **??** et 5Fig. 5(h):) par une soudaine remontée du nombre de clauses après une baisse régulière.

5 Conclusion

Il est maintenant clair que la mise à l'écart de la procédure originelle de Davis et Putnam a été trop rapide. Rien n'avait été en effet fait pour justifier celle-ci, mise à part une intuition de départ, remise en cause par les capacités mémoire des machines actuelles. De plus, alors que répondant à un problème bien plus difficile que DPL, très utile dans des contextes de compilation de base de connaissance, elle peut surpasser celle-ci sur un nombre relativement important d'exemples structurés. Or, l'intérêt de la course à la plus grande formule aléatoire traitée étant discutable d'un point de vue pratique, ce comportement de DP paraît intéressant dans le but de proposer de nouvelles méthodes pour la résolution de SAT.

Les contributions de cet article sont, en dehors de la première étude poussée de DP que nous présentons, d'en donner une implémentation exponentiellement plus rapide que la précédente proposée par Dechter et Rish. Ensuite nous montrons par l'expérimentation que DP est insensible au phénomène de transition de phase sur

3-cnf, mais plutôt qu'il dépend, lors d'un calcul de résolvant, du nombre de variables à éliminer, en montrant un net accroissement de difficulté autour de la moitié des variables à éliminer. De plus, nous montrons que DP donne des résultats très surprenants sur des instances de DIMACS en répondant à un problème plus difficile en moins de temps que les algorithmes basés sur DPL sans mémoire de conflits.

Les perspectives ouvertes par ce travail sont multiples. D'abord vient à l'idée une hybradation entre DP et DPL. En dehors de l'hybradation du type « *faire du DP tant qu'on peut puis faire du DPL* », on pourrait aussi basculer en DP lorsque l'affectation d'un littéral dans DPL conduit à seulement une chance sur deux d'obtenir un modèle contenant ce littéral. Il serait peut-être alors préférable de couper cette variable plutôt que de risquer de se tromper à 50%. Puis, on peut arguer que DP n'en est qu'à ses débuts, un peu comme DPL l'était il y a dix ans. Nous travaillons actuellement sur une structure de donnée de clauses adaptée à la fois aux tests et mises à jour de subsomptions mais aussi à l'opération de distribution, cœur de Calcres. Les résultats que nous commençons à obtenir sont encourageants et laisse penser qu'il reste encore des améliorations significatives possibles sur DP.

Références

[1]

[2] *? ? ?*, ? ? ?

[3] *sato3*, 1997.

[4] H. Kautz B. Selman and D. McAllester. Ten challenges in propositional reasoning and search. In *IJCAI-97*, 1997.

[5] B. Selman D. Mitchell and H. Levesque. Hard and easy distributions of sat problems. *AAAI*, 1990.

[6] M. Davis and H. Putnam. A computing procedure for quantification theory. *Journal of the ACM*, 1960.

[7] Johan de Kleer. An improved incremental algorithm for generating prime implicates. In *AAAI*, 1992.

[8] R. Dechter and I. Rish. Directional resolution : The davis-putnam procedure, revisited. In *Proceedings of KR-94*, pages 134–145, 1994.

[9] Olivier Dubois. Can a simple algithme could be efficient ? Disponible sur le web : ftp ://ftp.dimacs.rutgers.edu.

[10] J.-F. Héry and J.-C. Laleuf. Logique et modélisation : compatibilité entre données et modèles en logique des propositions. Technical report, EDF, Département Traitement de l'Information et Études Mathématiques, 1995.

[11] D. Johnson. A theorician's guide to the experimental analysis of algorithms. Disponible sur http ://www.research.att.com/ dsj/paper/exper.ps.

[12] G. Logeman M. Davis and D. Loveland. A machine program for theorem-proving. *Communications of the ACM*, 1962.

[13] R. Schrag and M. Crawford. Implicates and prime implicates in random 3-sat. *Artificial Intelligence*, 1996.

[14] Laurent Simon. Algorithmes de calcul de résolvant. Stage de DEA de l'université Orsay Paris XI.

[15] Colin P. Williams and Tad Hogg. The typicality of phase transitions in search. *Computational Intelligence*, 1993.

[16] Hantao Zhang and Mark E. Stickel. Implementing the davis-putnam algorithm by tries.