Où la polynomialité n'est qu'énumérative

Richard Génisson* — **Antoine Rauzy****

* *LIM, Université Aix-Marseille I,*
Technopôle de Château Gombert,
39, rue Joliot Curie,
13453 Marseille cedex 13, France
genisson@lim.univ-mrs.fr

** *LaBRI, Université Bordeaux I,*
351, cours de la Libération,
33405 Talence cedex, France,
rauzy@labri.u-bordeaux.fr

RÉSUMÉ. *Dans cet article, nous étudions l'efficacité des démonstrateurs généraux du type procédure de Davis & Putnam ou "Forward Checking" sur les principales restrictions polynomiales du problème SAT et des problèmes de satisfaction de contraintes. Pour ce faire, nous introduisons un schéma algorithmique paramétré de complexité polynomiale et nous en montrons la complétude sur les dites restrictions. Nous rendons compte d'expérimentations venant renforcer les résultats théoriques obtenus.*

ABSTRACT. *In this paper, we study the efficiency of general solvers such as the Davis-Putnam procedure or Forward Checking on the main polynomial restrictions of SAT and Constraint Satisfaction Problems. To achieve this goal, we introduce a parametrized algorithmic scheme of polynomial complexity. We show its completness on the former restrictions. We report experiments that enforce the obtained theoretical results.*

MOTS-CLÉS : *Problème SAT, Problèmes de Satisfaction de Contraintes, Algorithmes Énumératifs, Classes Polynomiales.*

KEYWORDS: *SAT Problem, Constraint Satisfaction Problems, Enumerative Algorithms, Polynomial Classes.*

1. Introduction

Parmi les problèmes NP-complets, le problème SAT et les problèmes de satisfaction de contraintes (CSP) occupent une place particulière pour au moins deux raisons. Tout d'abord, le problème SAT est le premier à avoir été démontré NP-complet [COO 71] et les CSP le généralisent. Ensuite, les formalismes sous-jacents sont suffisamment expressifs pour permettre de coder naturellement de nombreux problèmes, NP-complets ou non (allocation de ressources, preuve de circuits, analyse de risques...). L'utilisation de ces formalismes se heurte toutefois à l'inefficacité des algorithmes de traitement.

L'idéal serait évidemment de trouver des restrictions syntaxiques de ces deux formalismes qui, tout en conservant un bon pouvoir d'expression, permettraient de mettre en œuvre des méthodes de résolution efficaces, c'est-à-dire de complexité (faiblement) polynomiale. Malheureusement, les restrictions polynomiales connues à ce jour sont assez pauvres. Par exemple, aucun des problèmes réels évoqués plus haut ne s'exprime sous forme d'un ensemble de clauses de Horn ou de clauses binaires.

On peut dresser le tableau suivant de la situation. On dispose :

– d'un corpus de classes polynomiales plus ou moins naturelles et d'algorithmes de reconnaissance et de décision souvent subtils associés à ces classes. Des motivations essentiellement théoriques ont présidé au développement de ce corpus. Par exemple, trouver des algorithmes linéaires là où on ne connaissait que des algorithmes quadratiques. En général, ces algorithmes ne peuvent pas être étendus en des démonstrateurs généraux ;

– de démonstrateurs complets qui sont pour la plupart des variations sur le thème de Davis et Putnam dans le cas propositionnel ou du Forward Checking (ou MAC [SAB 94, BES 96]) dans le cas des CSP[1]. De nombreux travaux ont visé à améliorer l'efficacité pratique de ces démonstrateurs.

On peut bien sûr inclure dans les solveurs généraux des "briques algorithmiques" dédiées à la résolution efficace de tel ou tel problème particulier. Cela n'est cependant pas très satisfaisant. En effet, ces algorithmes spécialisés requièrent souvent des structures de données ad hoc. On obtient donc très vite des programmes difficiles à modifier et à maintenir. De plus, en pratique, l'appel régulier à des routines extérieures s'avère la plupart du temps plus pénalisant qu'efficace. D'où l'idée d'étudier la complexité des solveurs généraux sur les classes polynomiales connues, afin de voir si l'on a vraiment besoin de routines spécialisées pour garantir leur bon comportement sur ces classes.

Cette démarche pragmatique nous a conduits très vite à la conviction que les principales classes polynomiales du problème SAT se traitent très efficacement, tant d'un point de vue théorique que d'un point de vue expérimental à l'aide d'un schéma énumératif simple. Cet article présente donc deux types de résultats : une série de théo-

1. Dans [GÉN 96b], il a été montré que ces deux algorithmes sont en fait strictement équivalents.

rèmes établissant que la complexité au pire de ce schéma est faiblement polynomiale sur ces classes et des résultats expérimentaux montrant que ces performances sont excellentes en pratique.

Le reste de cet article comporte trois parties.

La section 2 présente le schéma algorithmique χ-$\mathtt{SAT}_{\mathtt{S},\mathtt{H},\mathtt{E}}$. Celui-ci est paramétré par une procédure de simplification S, une heuristique H, une procédure de test d'équivalence E et un nombre entier k. Ce dernier est la borne maximum du nombre de Strahler de l'arbre d'appels de la procédure. Le nombre de Stralher permet de caractériser la taille des arbres d'appels sans tenir compte du nombre de variables de la formule considérée. C'est donc un outil très puissant pour analyser l'efficacité des méthodes énumératives. Ainsi, le schéma χ-$\mathtt{SAT}_{\mathtt{S},\mathtt{H},\mathtt{E}}$ permet d'étudier la complexité d'une large classe de démonstrateurs sur différentes classes de formules. Nous montrons dans quelle mesure l'heuristique H peut être supprimée, dans le but de définir une hiérarchie $\aleph^{\star}_{\mathtt{S},\mathtt{E}}$ de classes de formules décidables en temps polynomial.

La section 3 est consacrée à la mise en œuvre du schéma χ-$\mathtt{SAT}_{\mathtt{S},\mathtt{H},\mathtt{E}}$. Nous y proposons donc une structure de données sur laquelle les procédures S, H et E peuvent être instanciées efficacement. La complexité du schéma sur les principales restrictions polynomiales du problème SAT (2-SAT, Horn-SAT, q-Horn SAT) y est étudiée en détails, démontrant que ce dernier est faiblement polynomial sur ces classes. Nous abordons aussi le problème du Horn renommage. Nous comparons ensuite la hiérarchie $\aleph^{\star}_{\mathtt{S},\mathtt{E}}$ à celle de Dalal et Etherington [DAL 92c]. Enfin, nous présentons quelques résultats expérimentaux montrant que le schéma χ-$\mathtt{SAT}_{\mathtt{S},\mathtt{H},\mathtt{E}}$ est très efficace en pratique, ici sur des instances 2-SAT.

Finalement, la section 4 montre que le schéma χ-$\mathtt{SAT}_{\mathtt{S},\mathtt{H},\mathtt{E}}$ est également efficace sur certaines classes polynomiales des CSP ou, plus précisément, sur le codage par des formules propositionnelles de ces problèmes. Nous y examinons les CSP arborescents et les CSP Zéro-Un-Tous pour deux codages différents (celui de de Kleer [KLE 89] et celui de Dalal [DAL 92b]).

2. Problème SAT, schémas énumératifs et polynomialité

Cette section présente le schéma énumératif χ-$\mathtt{SAT}_{\mathtt{S},\mathtt{H},\mathtt{E}}$ et discute des conditions qui le rendent polynomial.

2.1. *Rappels sur le problème SAT*

Une formule propositionnelle est sous forme normale conjonctive (CNF) si elle est de la forme $C_1 \wedge C_2 \wedge \ldots \wedge C_m$, où chaque clause C_i est une disjonction de littéraux et chaque littéral est soit une variable x_i soit sa négation $\neg x_i$. On note $\neg p$ ou $\bar{p}$ l'opposé du littéral p. On assimile les formules CNF à des ensembles de clauses et les clauses à des ensembles de littéraux. On note $\emptyset$ l'ensemble vide de clauses et $\Box$ la

clause vide. On note respectivement $var(\phi)$ et $lit(\phi)$ l'ensemble des variables et des littéraux apparaissant dans la formule ϕ.

Le problème SAT consiste à déterminer s'il existe une affectation de valeurs booléennes aux variables rendant vraie la formule CNF considérée. Cook [COO 71] a montré que le problème SAT est NP-complet et c'est depuis un des problèmes NP-complets de référence [GAR 79]. Une formule CNF est dite satisfiable si une telle affectation existe et insatisfiable sinon.

Une formule CNF dont toutes les clauses contiennent k littéraux est dite k-CNF. Il est facile de montrer que toute formule CNF peut être transformée en une formule 3-CNF équivalente pour la satisfiabilité [GAR 79].

On connaît quelques restrictions du problème SAT pour lesquelles on dispose d'algorithmes de reconnaissance et décision de complexité polynomiale, voire même linéaire pour Horn-SAT – satisfiabilité d'ensembles de clauses de Horn – et 2-SAT – satisfiabilité d'ensembles de clauses binaires – (voir [KLE 99] pour une vue d'ensemble sur la question).

Soient ϕ un ensemble de clauses, $x_1, \ldots, x_n$ des variables apparaissant dans ϕ et $v_1, \ldots, v_n$ des valeurs. On note $\phi_{x_1 \leftarrow v_1, \ldots, x_n \leftarrow v_n}$ l'ensemble ϕ dans lequel les valeurs $v_1, \ldots, v_n$ ont été simultanément substituées (ou affectées) aux variables $x_1, \ldots, x_n$. $\phi_{x_1 \leftarrow v_1, \ldots, x_n \leftarrow v_n}$ est équivalent à l'ensemble obtenu à partir de ϕ en supprimant les clauses contenant un littéral satisfait et en supprimant les littéraux falsifiés des autres clauses. Toute affectation partielle $A : x_1 \leftarrow v_1, \ldots, x_n \leftarrow v_n$ peut être assimilée à un ensemble de littéraux : $A = \{x_i; v_i = 1\} \cup \{\neg x_i; v_i = 0\}$. Réciproquement, tout ensemble de littéraux ne contenant pas un littéral et son opposé peut être assimilé à une affectation partielle. On note $A(x)$ la valeur de la variable x dans l'affectation partielle A.

Un littéral p est dit *unitaire* dans une formule ϕ s'il existe une clause de ϕ dont p est l'unique élément. Il est dit *pur* si son opposé n'apparaît pas dans ϕ.

2.2. *Schémas énumératifs pour résoudre SAT*

Nous appelons procédure énumérative pour résoudre SAT toute procédure fonctionnant suivant le schéma algorithmique paramétré suivant :

```
0: SAT_{S,H,E}(φ)
1:     ψ ← S(φ)                          simplification
2:     si ψ = ∅ retourner 1
3:     si □ ∈ ψ retourner 0
4:     p ← H(ψ)                          heuristique
5:     v ← SAT_{S,H,E}(ψ_p)
6:     si E(ψ,ψ_p,v) retourner v         retour arrière intelligent
7:     retourner SAT_{S,H,E}(ψ_¬p)
```

Dans ce schéma,

– ϕ et ψ dénotent des ensembles de clauses, p un littéral et v une valeur (0 ou 1, c.à.d. satisfiable ou non satisfiable) ;

– S est une procédure de simplification de l'ensemble de clauses. Un choix raisonnable pour S est la propagation unitaire UP qui consiste à propager les littéraux unitaires et purs. UP peut se décrire de la façon suivante :

```
0:  UP(φ)
1:      tant que φ contient un littéral p unitaire ou pur faire
2:          φ ← φ_p
3:      fait
4:      retourner φ
```

Nous verrons section 3 que UP est linéaire en la taille de la formule, pourvu que l'on choisisse les bonnes structures de données.

– H est une heuristique choisissant un littéral apparaissant dans l'ensemble de clauses. De nombreuses heuristiques ont été proposées dans la littérature (voir par exemple [HOO 95] pour une discussion sur ces dernières) ;

– E est une procédure qui détermine si la formule ψ est équivalente, du point de vue de la satisfiabilité, à la formule ψ_p. Nous avons appelé E retour arrière intelligent parce que si cette équivalence est établie, il est inutile de considérer la formule $\psi_{\neg p}$. C'est typiquement le cas si $v = 1$. Un choix raisonnable pour E consiste à appliquer le principe d'autark (quand la valeur retournée v est 0). Le principe d'autark [MON 85], appelé partition du modèle dans [JEA 88], s'énonce comme suit :

Soit ϕ une formule et p_1,...,p_k des littéraux de ϕ. Si $\mathtt{UP}(\phi_{p_1,\ldots,p_k}) \subset \phi$, alors ϕ est satisfiable si et seulement si $\mathtt{UP}(\phi_{p_1,\ldots,p_k})$ l'est.

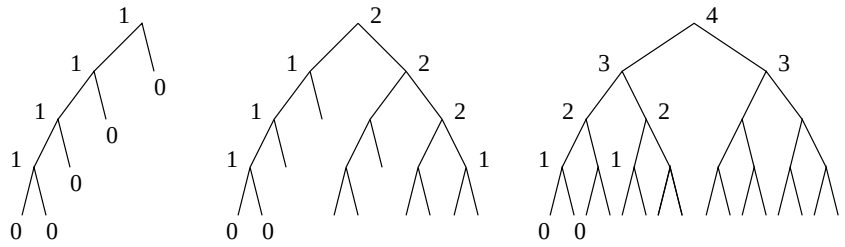
On remarquera que si UP provoque l'apparition de la clause vide, on ne peut pas avoir $\mathtt{UP}(\phi_{p_1,\ldots,p_k}) \subset \phi$. Ce principe généralise la notion de littéral pur. Nous verrons section 3 que les cas d'autark peuvent être détectés en temps linéaire pourvu que l'on choisisse les structures de données adéquates. Dans la suite, nous appelerons AUTARK la procédure mettant en œuvre ce principe.

Le schéma $\mathtt{SAT}_{\mathtt{S},\mathtt{H},\mathtt{E}}$ est correct et complet. Il dépend implicitement d'un quatrième paramètre : la structure de données utilisée pour coder les clauses. Il englobe un grand nombre de démonstrateurs proposés dans la littérature. En particulier, la fameuse procédure de Davis et Putnam [DAV 62] peut être vue comme l'instanciation de ce schéma avec UP comme procédure de simplification et un retour arrière intelligent réduit au seul cas où $v = 1$.

2.3. *Schémas énumératifs et polynomialité*

La question suivante est au cœur de cet article : soit Φ une classe de formules CNF, le schéma $\mathtt{SAT}_{\mathtt{S},\mathtt{H},\mathtt{E}}$ est-il de complexité polynomiale sur Φ ?

Dans ce qui suit, on supposera toujours que la taille des formules considérées est bornée par un polynôme du nombre de variables apparaissant dans ces formules.



(a) l'arbre UP (b) l'arbre 2SAT (c) l'arbre binaire complet

Figure 1. *Trois arbres binaires typiques et leur nombre de Stralher.*

On choisira, comme usuellement, ce nombre comme mesure de la taille des formules [GAR 79]. On supposera de plus que les procédures `S`, `H`, `E` ainsi que le calcul de ϕ_p et $\phi_{\neg p}$ sont de complexités polynomiales.

Tester la satisfiabilité d'une formule à l'aide de $\mathtt{SAT}_{\mathtt{S,H,E}}$ revient à parcourir implicitement un arbre binaire, chaque nœud correspondant à un appel de la procédure. Compte tenu des suppositions faites plus haut, $\mathtt{SAT}_{\mathtt{S,H,E}}$ est de complexité polynomiale sur Φ s'il existe un entier k tel que pour toute formule ϕ de Φ la taille de l'arbre des appels de $\mathtt{SAT}_{\mathtt{S,H,E}}$ sur ϕ est en $\mathcal{O}(V^k)$, où V dénote le nombre de variables sur lesquelles ϕ est construite.

Afin d'établir des résultats intéressants, il est pratique de se débarrasser du nombre de variables des formules considérées. On utilise pour cela le nombre de Strahler χ qui caractérise la forme des arbres binaires (voir [VIE 90] pour une introduction à cette notion et à ses multiples applications). Soit B un arbre binaire. $\chi(B)$ est défini comme suit :

– si B est une feuille alors $\chi(B) = 0$;
– si B est de la forme $\langle B_1, B_2 \rangle$ alors

- si $\chi(B_1) = \chi(B_2)$, alors $\chi(B) = \chi(B_1) + 1 = \chi(B_2) + 1$,
- sinon $\chi(B) = max(\chi(B_1), \chi(B_2))$.

La figure 1 illustre l'idée de Stralher : l'arbre (a), qui correspond à l'arbre d'appel d'une procédure de propagation unitaire, a une mesure de Stralher de 1 indépendamment du nombre de variables. L'arbre (b) qui est déjà un peu plus touffu et qui correspond à l'arbre d'appel d'une procédure énumérative (du type $\mathtt{SAT}_{\mathtt{S,H,E}}$ sur des instances 2SAT, a une mesure de Stralher de 2 indépendamment du nombre de variables. Finalement, l'arbre (c), qui est l'arbre binaire complet, a une mesure de Stralher égale à la profondeur de l'arbre.

On note $|\mathtt{SAT}_{\mathtt{S,H,E}}(\phi)|$ la taille de l'arbre des appels de $\mathtt{SAT}_{\mathtt{S,H,E}}$ sur ϕ (c'est-à-dire le nombre d'appels récursifs de la procédure) et $\chi(\mathtt{SAT}_{\mathtt{S,H,E}}(\phi))$ le nombre de Strahler de cet arbre.

Soient k_1 et k_2 deux entiers, par abus on notera $\chi(k_1,k_2)$ le nombre k tel que $k = k_1 + 1 = k_2 + 1$ si $k_1 = k_2$ et $max(k_1,k_2)$ sinon.

Soit $\alpha(V,k)$ le nombre maximum de nœuds d'un arbre binaire de profondeur maximum V et de nombre de Stralher k. On a : $\alpha(V,k) \leq \alpha(V-1,k) + \alpha(V-1,k-1) \leq \sum_{i=1}^{V} \alpha(V-i,k-1) \leq V \times \alpha(V-1,k-1)$. Par conséquent, on prouve facilement par induction que $\alpha(V,k) \leq V^k$.

Le lemme suivant applique ce résultat à la taille de l'arbre d'appel du schéma.

Lemme 1 (Nombre de Strahler et polynomialité) *Soit ϕ une formule contenant V variables et soit k un entier positif. Si $\chi(\mathtt{SAT}_{\mathtt{S},\mathtt{H},\mathtt{E}}(\phi)) = k$ alors $|\mathtt{SAT}_{\mathtt{S},\mathtt{H},\mathtt{E}}(\phi)|$ est en $\mathcal{O}(V^k)$.*

Par conséquent, pour démontrer la polynomialité de $\mathtt{SAT}_{\mathtt{S},\mathtt{H},\mathtt{E}}$ sur Φ, il suffit de prouver qu'il existe un entier k tel que pour toute formule ϕ de Φ, on a $\chi(\mathtt{SAT}_{\mathtt{S},\mathtt{H},\mathtt{E}}(\phi)) \leq k$. On modifie donc le schéma algorithmique comme suit :

```
0:   χ-SAT_{S,H,E}(φ,k_max)
1:     ψ ← S(φ)
2:     si ψ = ∅ retourner ⟨1,0⟩
3:     si □ ∈ ψ retourner ⟨0,0⟩
4:     si k_max = 0 retourner ⟨?,0⟩                          incomplétude
5:     p ← H(ψ)
6:     ⟨v,k_1⟩ ← χ-SAT_{S,H,E}(ψ_p,k_max)
7:     si E(ψ,ψ_p,v) retourner ⟨v,χ(k_1,0)⟩
8:     si k_1 = k_max ⟨v,k_2⟩ ← χ-SAT_{S,H,E}(ψ_¬p,k_max − 1)   contrôle de χ
9:     sinon ⟨v,k_2⟩ ← χ-SAT_{S,H,E}(ψ_¬p,k_max)
10:    retourner ⟨v,χ(k_1,k_2)⟩
```

Ce nouveau schéma prend un paramètre supplémentaire : k_{max}, la borne supérieure que l'on s'alloue pour le nombre de Strahler de l'arbre des appels. Il est donc incomplet (ligne 4). Il retourne un couple de valeurs : la satisfiabilité v (0, 1 ou ?) et le nombre de Strahler k de l'arbre construit. k sert à décrémenter k_{max} lorsque le nombre de Strahler de la branche gauche atteint la borne fixée (ligne 8).

Le schéma χ-$\mathtt{SAT}_{\mathtt{S},\mathtt{H},\mathtt{E}}$ peut être vu de deux façons. Comme un outil théorique : pour prouver la polynomialité du schéma $\mathtt{SAT}_{\mathtt{S},\mathtt{H},\mathtt{E}}$ sur une classe Φ, il suffit de prouver qu'il existe un entier k pour lequel χ-$\mathtt{SAT}_{\mathtt{S},\mathtt{H},\mathtt{E}}(.,k)$ est complet. Comme un outil pratique : en tant qu'algorithme incomplet mais de complexité bornée.

2.4. *Classes polynomiales pour les schémas énumératifs*

Le schéma χ-$\mathtt{SAT}_{\mathtt{S},\mathtt{H},\mathtt{E}}$ permet de caractériser les classes de formules sur lesquelles le schéma $\mathtt{SAT}_{\mathtt{S},\mathtt{H},\mathtt{E}}$ est de complexité polynomiale. On peut ainsi définir une hiérarchie $\aleph_{\mathtt{S},\mathtt{H},\mathtt{E},(n)}$ de classes de formules : une formule ϕ appartient au niveau $\aleph_{\mathtt{S},\mathtt{H},\mathtt{E},k}$ de la

hiérarchie si χ-$\mathtt{SAT}_{\mathtt{S},\mathtt{H},\mathtt{E}}(.,k)$ décide ϕ. Le test de reconnaissance (d'appartenance au niveau) est ici confondu avec l'algorithme de décision (pour la satisfiabilité).

Les heuristiques créent une asymétrie fondamentale entre les formules satisfiables et les formules insatisfiables. En effet, en supposant qu'on dispose d'une heuristique parfaite (c.à.d. faisant toujours "le" bon choix), le nombre de Stralher de l'arbre des appels est toujours 1 pour les formules satisfiables, alors qu'il peut prendre une valeur arbitrairement grande (mais toujours inférieure au nombre de variables) pour les formules insatisfiables. Cependant, les heuristiques dont on dispose en pratique sont mauvaises : comme le remarquent fort justement Hooker et Vinay [HOO 95], on n'observe pas d'écart de performances énorme suivant que l'on traite des formules satisfiables ou insatisfiables. Les procédures énumératives passent donc le plus clair de leur temps à démontrer l'insatisfiabilité des (sous-)formules considérées, même lorsque la formule principale est satisfiable. Quoi qu'il en soit, il est intéressant de définir une hiérarchie de classes de formules qui soit indépendante de l'heuristique choisie. Cela nous conduit à utiliser un nouveau schéma énumératif :

0: χ-$\mathtt{SAT}^{\star}_{\mathtt{S},\mathtt{E}}(\phi,k_{max})$
1: $\psi \leftarrow \mathtt{S}(\phi)$
2: `si` $\psi = \emptyset$ `retourner` $\langle 1,0\rangle$
3: `si` $\Box \in \psi$ `retourner` $\langle 0,0\rangle$
4: `si` $k_{max} = 0$ `retourner` $\langle ?,0\rangle$
5: trouvé $\leftarrow$ faux, $L \leftarrow lit(\psi)$, $k^{\star} \leftarrow 0$
6: `tant que` non trouvé `et` $L \neq \emptyset$ `faire`
7: extraire un littéral p de L
8: $\langle v,k\rangle \leftarrow \chi$-$\mathtt{SAT}^{\star}_{\mathtt{S},\mathtt{E}}(\psi_p,k_{max}-1)$
9: `si` $v \in \{0,1\}$ `alors` trouvé $\leftarrow$ vrai
10: `si` $k > k^{\star}$ `alors` $k^{\star} \leftarrow k$
11: `fait`
12: `si` non trouvé `retourner` $\langle ?,\chi(k^{\star},0)\rangle$
13: `si` $\mathtt{E}(\psi,\psi_p,v)$ `retourner` $\langle v,\chi(k^{\star},0)\rangle$
14: $\langle v,k\rangle \leftarrow \chi$-$\mathtt{SAT}^{\star}_{\mathtt{S},\mathtt{E}}(\psi_{\neg p},k_{max})$
15: `retourner` $\langle v,\chi(k^{\star},k)\rangle$

L'idée du schéma χ-$\mathtt{SAT}_{\mathtt{S},\mathtt{E}}$ est illustrée sur la figure 2. La boucle permet de rechercher un sous-arbre gauche complet pour la formule et de nombre de Stralher au plus $k_{max}-1$. L'arbre construit a donc comme nombre de Stralher au plus k_{max} et sa profondeur est au plus $V.(V-1)$. Il est donc facile de vérifier que le nombre maximum d'appels $\alpha^{\star}(V,k)$ de χ-$\mathtt{SAT}^{\star}_{\mathtt{S},\mathtt{E}}(.,k)$ sur une formule contentant V variables est en $\mathcal{O}(V^{2k})$: on a $\alpha^{\star}(V,k) \leq 2V \times \alpha^{\star}(V-1,k-1)+\alpha^{\star}(V-1,k) \leq V^2 \times \alpha^{\star}(V-1,k-1)$. □

L'intérêt du schéma χ-$\mathtt{SAT}^{\star}_{\mathtt{S},\mathtt{E}}$ tient à ce que, pour toute formule insatisfiable ϕ, pour toute heuristique $\mathtt{H}$ et pour tout entier k, si χ-$\mathtt{SAT}_{\mathtt{S},\mathtt{H},\mathtt{E}}(.,k)$ décide ϕ, alors χ-$\mathtt{SAT}^{\star}_{\mathtt{S},\mathtt{E}}(.,k)$ décide aussi ϕ. χ-$\mathtt{SAT}^{\star}_{\mathtt{S},\mathtt{E}}$ permet donc de se débarrasser des heuristiques au prix d'une augmentation polynomiale (en fait quadratique) de la complexité. Cela

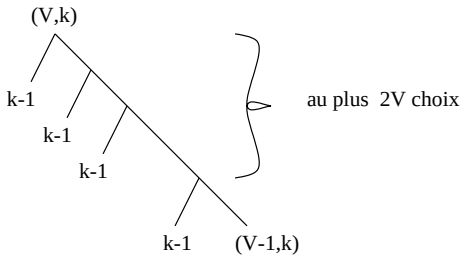


Figure 2. *Illustration du schéma* $\mathtt{SAT}^{\star}_{\mathtt{S,E}}$ *: les branches gauches correspondent aux appels récursifs effectués dans la boucle.*

n'a vraiment de sens que pour les formules insatisfiables car, pour les formules satisfiables, les heuristiques permettent éventuellement de "descendre" directement sur la solution.

On définit donc la hiérarchie $\aleph^{\star}_{\mathtt{S,E},(n)}$ de classes de formules comme suit : une formule ϕ appartient au niveau $\aleph^{\star}_{\mathtt{S,E},k}$ si χ-$\mathtt{SAT}^{\star}_{\mathtt{S,E}}(.,k)$ décide ϕ.

La hiérarchie $\aleph^{\star}_{\mathtt{UP,AUTARK},(n)}$ est puissante et raisonnable. Puissante, car elle capture les principales classes polynomiales du problème SAT et des problèmes de satisfaction de contraintes (voir sections 3 et 4). Raisonnable, car elle se décrit par un schéma algorithmique simple qui est susceptible d'être utilisé en tant que démonstrateur généraliste. Les liens entre cette hiérarchie et celle de Dalal & Etherington [DAL 92c] sont explorés section 3.9.

Pour finir, on remarquera que le schéma χ-$\mathtt{SAT}^{\star}_{\mathtt{S,E}}$ n'élimine pas complétement l'intérêt des heuristiques dans la mesure où il faut bien choisir l'ordre dans lequel on teste les littéraux de la formule considérée (ligne 7).

3. Classes Polynomiales du problème SAT

La complexité de la procédure de Davis et Putnam sur les deux principales classes polynomiales du problème SAT – Horn-SAT et 2-SAT – a été analysée en détails dans [RAU 95]. Dans cette section, Nous réinterprétons (et nous étendons) ces résultats dans le cadre du schéma algorithmique χ-$\mathtt{SAT}_{\mathtt{S,H,E}}$ appliqué aux ensembles de clauses de cardinalités. Puis, nous appliquons le schéma au problème du Horn-renommage et aux q-Horn clauses. Nous comparerons enfin la hiérarchie $\aleph^{\star}_{S,E}$ à celle de Dalal et Etherington [DAL 92c]. Notre objectif est de démontrer qu'à condition de bien choisir les structures de données, le schéma χ-$\mathtt{SAT}_{\mathtt{S,H,E}}$ est un outil algorithmique puissant et adaptable et qu'il est superbement efficace en pratique.

3.1. *Clauses de cardinalité*

Le schéma $\mathtt{SAT}_{\mathtt{S,H,E}}$ peut être mis en œuvre sur des formalismes plus généraux que les ensembles de clauses (voir par exemple [DAL 92a, DAL 92b, RAU 99]). Dans cet article, nous nous intéressons à une légère extension du formalisme. Cette extension consiste à considérer des ensembles de formules (ou clauses) de cardinalité plutôt que de clauses classiques. Elle permet une expression plus compacte de certains problèmes et ne demande pour ainsi dire aucun changement de structures de données.

Une formule de cardinalité est un triplet (b,h,L) dans lequel b et h sont des entiers et L un ensemble de littéraux. Une telle formule est satisfaite quand au moins b et au plus h des littéraux de L sont satisfaits. Une clause classique est donc une clause de cardinalité de la forme $(1,|L|,L)$, où $|L|$ dénote la taille de la liste L. Dans ce qui suit, on supposera implicitement que L ne contient pas un littéral et son opposé ni deux occurrences du même littéral (ce qui permet de parler de L comme d'une liste), que $b \leq h$, et qu'au moins une des inégalités $b > 0$ et $h < |L|$ est vérifiée.

La clause (b,h,L) est unitaire si $b = |L|$ ou si $h = 0$. Dans le premier cas, tous les littéraux de L doivent être satisfaits (pour que la clause soit satisfaite). Dans le second, ils doivent tous être falsifiés. Un littéral est unitaire dans un ensemble de clauses ϕ s'il apparaît dans une clause unitaire de ϕ.

La clause (b,h,L) est monotone négative si $b = 0$. Elle est monotone positive si $h = |L|$. Elle est monotone si elle est monotone positive ou monotone négative. L'idée est que si $b = 0$ (resp. $h = |L|$) falsifier (resp. satisfaire) un littéral de L ne fait que renforcer la satisfaction de la clause. Un littéral p est pur dans une formule ϕ s'il n'apparaît que dans des clauses monotones positives et que son opposé n'apparaît que dans des clauses monotones négatives de ϕ. En fait, la clause $(b,h,\{p_1,\ldots,p_k\})$ est équivalente à la clause $(|L|-h,|L|-b,\{\neg p_1,\ldots,\neg p_k\})$, si bien que les deux conditions n'en font qu'une.

Soit A une affectation partielle des variables apparaissant dans une formule de cardinalité (b,h,L). $(b,h,L)_A$ est équivalente à la clause $(b-s,h-s,N)$ où s dénote le nombre de littéraux de L satisfaits par A et N dénote le sous-ensemble des littéraux de L qui ne sont pas affectés par A (ces derniers sont neutres pour A). Pour des raisons qui apparaîtront bientôt, il est intéressant de définir le statut d'une clause sous une affectation partielle en gardant tous les littéraux de la clause. Soient donc respectivement s, f et n, les nombres de littéraux de L satisfaits, falsifiés et neutres pour A. On dit que :

– (b,h,L) est inchangée par A, si $n = |L|$;

– (b,h,L) est satisfaite par A, si $s \geq b$ et $s + n \leq h$;

– (b,h,L) est falsifiée par A, si $s + n < b$ ou si $s > h$;

– (b,h,L) est affaiblie par A, si $b = 0$, $s = 0$ et $f > 0$ ou si $h = |L|$, $f = 0$ et $n > 0$;

– finalement, (b,h,L) est raccourcie dans les autres cas.

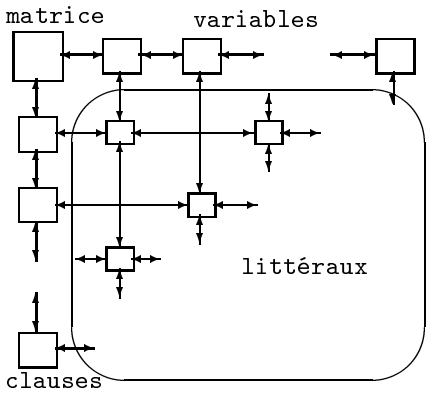


Figure 3. *Une matrice creuse*

Le lemme suivant généralise la notion d'autark.

Lemme 2 (Autark généralisé) *Soient ϕ un ensemble de clauses de cardinalité et A une affectation partielle de variables de ϕ. Si ϕ_A ne contient ni clauses falsifiées, ni clauses raccourcies par A, alors ϕ est satisfiable si et seulement si ϕ_A est satisfiable.*

Supposons ϕ satisfiable et B une affectation (totale) des variables de ϕ telles que $\phi_B = 1$. Soit C l'affectation définie comme suit : $C(x) = A(x)$ si A donne une valeur à x et $C(x) = B(x)$ sinon. Par définition C satisfait les clauses satisfaites et inchangées par A. Il est de plus facile de vérifier que toute clause affaiblie par A est satisfaite par C. □

3.2. *Matrices creuses*

Afin de mettre en œuvre le schéma $\mathtt{SAT}_{\mathtt{S,H,E}}$ (appliqué aux ensembles de clauses de cardinalité), la meilleure structure de donnée est sans doute la matrice creuse. Chaque ligne de la matrice code une clause (c'est-à-dire une liste de littéraux). Chaque colonne code la liste des occurrences d'une variable. Cette structure est schématisée sur la figure 3.

Pour mettre en œuvre un algorithme énumératif sur une telle structure de données, la principale opération est l'affectation d'une valeur appartenant à {*vrai, faux, neutre*} à une variable. Cette opération est effectuée en visitant chaque occurrence de la variable pour mettre à jour différents compteurs.

– Pour chaque clause, on maintient des compteurs du nombre de littéraux satisfaits, falsifiés et neutres qu'elle contient, ainsi que son statut (inchangée, satisfaite, falsifié, affaiblie ou raccourcie).

– On maintient un compte global du nombre de clauses de chaque catégorie.

– On maintient la valeur de chaque variable ainsi que les comptes de ses occurrences positives et négatives dans les clauses de chaque statut et de ses occurrences positives et négatives dans les clauses unitaires et dans les clauses monotones. Si une variable apparaît à la fois positivement et négativement dans les clauses unitaires, l'ensemble de clauses courant est insatisfiable.

– On maintient une liste des littéraux unitaires et une liste des littéraux purs. Ces deux listes sont doublement chaînées pour permettre l'insertion et le retrait en temps constant.

Cette façon d'effectuer les affectations n'est pas celle préconisée habituellement (par exemple dans [KLE 99]). En général, les auteurs supposent que l'ensemble de clauses est mis à jour chaque fois qu'une affectation est effectuée.

Dans la suite, nous désignerons par C le nombre de clauses, V le nombre de variables, et S la taille de la donnée, c'est-à-dire le nombre de littéraux apparaissant dans l'ensemble de clauses considéré.

3.3. *Résolution unitaire linéaire et Autark*

La structure de matrice creuse permet une mise en œuvre linéaire de la propagation unitaire UP (voir section 2.2).

Lemme 3 (Propagation unitaire) *Soient ϕ un ensemble de clauses de cardinalité et $l_1, \ldots, l_k$ des littéraux. Le calcul de $\mathtt{UP}(\phi_{l_1,\ldots,l_k})$ est en $\mathcal{O}(S)$ sur la structure de matrice creuse.*

Puisqu'on maintient les listes des littéraux unitaires et purs, la détection d'un tel littéral est faite en temps constant. Considérons le nombre maximum de fois où un littéral est visité lors du calcul de $\mathtt{UP}(\phi_{l_1,\ldots,l_k})$. Une clause peut changer au plus trois fois de statut durant la propagation (de inchangée à affaiblie, de affaiblie à raccourcie, et finalement de raccourcie à falsifiée ou à satisfaite). Par conséquent, un littéral sera visité au plus trois fois "horizontalement" pour mettre à jour les compteurs des variables correspondantes. À ces trois fois, il faut éventuellement ajouter une quatrième lorsqu'on doit détecter les littéraux neutres d'une clause devenant unitaire et une cinquième pour mettre à jour les compteurs servant à détecter les littéraux purs. La liste des occurrences d'une variable est parcourue "verticalement" au plus une fois, quand la variable reçoit une valeur. Il s'ensuit que chaque littéral est visité au plus six fois lors du calcul de $\mathtt{UP}(\phi_{l_1,\ldots,l_k})$. □

On remarquera de plus que ce coût est le même que l'on fasse d'abord toutes les affectations des l_i puis une propagation des littéraux unitaires et purs ou que l'on fasse suivre chaque affectation d'un l_i par une telle propagation. Pour plus de détails bibliographiques sur la propagation des littéraux unitaires, le lecteur pourra consulter [GÉN 96a].

L'intérêt du lemme 3 est qu'il donne le coût d'un algorithme énumératif construisant un arbre de décision comportant B branches : un tel algorithme est en $\mathcal{O}(B \times S)$ (par branche on entend toute séquence d'affectations s'étendant de la racine à une feuille de l'arbre).

Pour mettre en œuvre efficacement la détection d'autark, on associe un nouveau compteur à chaque variable et à chaque clause : la date à laquelle une variable a reçu une valeur, respectivement la clause a changé de statut. Cette date est incrémentée à chaque appel de la procédure $\texttt{SAT}_{\texttt{S,H,E}}$. La mise à jour des dates est faite lors de la propagation unitaire sans surcoût.

Soit ϕ_1 la formule considérée à un certain nœud de l'arbre des appels de $\texttt{SAT}_{\texttt{S,H,E}}$ et soit ϕ_2 celle considérée à un nœud descendant de ce premier nœud. Soient d_1 et d_2 les dates associées respectivement à ces deux nœuds ($d_1 < d_2$). Alors, il y a autark entre ϕ_1 et ϕ_2 si et seulement si ϕ_2 ne contient aucune clause raccourcie de date d_3 telle que $d_1 \leq d_3 < d_2$ [JEA 88].

Par conséquent, la détection des cas d'autark peut être faite sans coût additionnel lors de la mise à jour de la matrice creuse.

3.4. *Heuristiques*

En pratique, une bonne heuristique pour choisir la prochaine variable à affecter améliore considérablement les performances des algorithmes énumératifs. Un compromis doit être trouvé entre le coût de l'heuristique et son efficacité. En fait, trois types d'heuristiques sont en général considérés.

Les heuristiques rapides (en $\mathcal{O}(1)$). Dans cette première catégorie, on trouve l'heuristique appelée `BIMO` (pour "Best In Matrix Order") dans [RAU 95] et qui consiste à choisir la première variable neutre apparaissant dans la matrice et la valeur satisfaisant le littéral le plus fréquent. Le fait que cette heuristique a un coût constant est exploité dans l'article cité.

La deuxième catégorie est constituée des heuristiques cherchant le littéral maximisant un certain critère en parcourant la liste des variables et en utilisant les différents compteurs. Ces heuristiques sont en $\mathcal{O}(V)$. Dans [RAU 95], on propose par exemple de choisir la variable qui est : 1) la plus fréquente dans les clauses raccourcies et 2) la plus fréquente dans les clauses inchangées, la valeur choisie étant celle satisfaisant le littéral le plus fréquent. Cette heuristique, appelée `FFIS` (pour "First Fail in Shortened Clauses") dans [RAU 95], est proche d'autres heuristiques proposées dans la littérature (voir par exemple [DUB 96, HOO 95, CRA 96]).

La troisième catégorie est constituée des heuristiques "regardant en avance" à l'aide de la propagation unitaire. L'idée est de détecter les cas où il existe un littéral p tel que $\texttt{UP}(\phi_p)$ contient la clause vide. Pour cela, il suffit de calculer $\texttt{UP}(\phi_x)$ et $\texttt{UP}(\phi_{\neg x})$ pour toutes les variables x apparaissant positivement et négativement dans les clauses binaires. De plus, si, pour un certain littéral p, le calcul de $\texttt{UP}(\phi_p)$ ne pro-

duit pas la clause vide, alors aucune des affectations effectuées lors de ce calcul ne peut la produire. On peut donc retirer les littéraux correspondants de la liste des littéraux candidats au calcul. Si aucun cas d'insatisfiabilité n'a été détecté, on peut utiliser les calculs ainsi effectués pour choisir le littéral et/ou appliquer une heuristique de la deuxième catégorie. Les meilleurs démonstrateurs actuels utilisent ce type d'heuristiques [DUB 96, CRA 96, LI 97], que nous désignerons par le terme générique UPLA (pour "Unit Propagation Look Ahead"). Les heuristiques UPLA sont en $\mathcal{O}(V \times S)$.

Dans la suite, nous utiliserons les heuristiques BIMO, FFIS et UPLA en tant que représentants de leurs catégories respectives. Nous désignerons par $\mathcal{H}$ la complexité d'une heuristique H.

On peut donc établir le théorème suivant.

Théorème 4 (Complexité du schéma $\mathtt{SAT}_{\mathtt{UP,H,AUTARK}}$)
Si le schéma χ-$\mathtt{SAT}_{\mathtt{UP,H,AUTARK}}(.,k)$ est complet sur une classe Φ de formules, alors le schéma $\mathtt{SAT}_{\mathtt{UP,H,AUTARK}}$ est en $\mathcal{O}(V^{k-1} \times (\mathcal{H} + S))$ sur Φ.

En particulier, les schémas $\mathtt{SAT}_{\mathtt{UP,BIMO,AUTARK}}$, $\mathtt{SAT}_{\mathtt{UP,FFIS,AUTARK}}$ et $\mathtt{SAT}_{\mathtt{UP,UPLA,AUTARK}}$ sont respectivement en $\mathcal{O}(V^{k-1} \times S)$, $\mathcal{O}(V^{k-1} \times S)$ et $\mathcal{O}(V^{k} \times S)$.

3.5. *Horn-SAT, 2-SAT*

Un ensemble de clauses (classiques) est de Horn si ses clauses contiennent au plus un littéral positif. Il est Horn renommable, s'il existe un renommage le transformant en un ensemble de clauses de Horn (un renommage remplace, pour certaines variables x, les occurrences de x par $\neg x$ et réciproquement).

Théorème 5 (Horn-SAT) *Le schéma χ-$\mathtt{SAT}_{\mathtt{UP,H,AUTARK}}(.,2)$ est complet pour les ensembles de clauses Horn renommables.*

La propagation des littéraux unitaires suffit pour détecter qu'un ensemble de clauses Horn renommable est insatisfiable. Si l'ensemble ϕ considéré est satisfiable, l'algorithme choisit un littéral p, et deux cas sont possibles. Si ϕ_p est insatisfiable, on va le détecter par propagation des littéraux unitaires. On revient donc en arrière, et on rappelle la procédure sur $\phi_{\neg p}$ et 2 (comme borne du nombre de Strahler). Si ϕ_p est satisfiable, il n'est pas nécessaire de rappeler la procédure sur $\phi_{\neg p}$. □

Une clause de cardinalité (b,h,L) est simple si elle est unitaire, ou si $b = 0$ et $h = 1$ ou si $b = |L| - 1$ et $h = |L|$. On peut établir un théorème similaire au précédent pour les ensembles de clauses de cardinalité simples.

Théorème 6 (Clauses simples) *Le schéma χ-$\mathtt{SAT}_{\mathtt{UP,H,AUTARK}}(.,2)$ est complet pour les ensembles de clauses de cardinalité simples.*

Soit ϕ un ensemble de clauses de cardinalité simples. On peut supposer sans perte de généralité que ϕ ne contient ni littéral unitaire, ni littéral pur. Soit p le littéral choisi par l'heuristique H. Deux cas sont possibles. Si $\mathtt{UP}(\phi_p)$ contient une clause vide, on revient en arrière, et on rappelle la procédure sur $\phi_{\neg p}$ et 2. Si $\mathtt{UP}(\phi_p)$ ne contient pas de clause vide, alors on est dans un cas d'autark. En effet, une clause simple raccourcie est soit unitaire soit vide. On n'a donc pas besoin de rappeler l'algorithme sur $\phi_{\neg p}$. □

Les clauses (classiques ou non) binaires sont simples ou de la forme $(1,1,[p,q])$. Dans ce dernier cas aussi, les clauses sont satisfaites ou falsifiées après propagation unitaire. D'où le corollaire suivant.

Corollaire 7 (2-SAT) *Le schéma* χ-$\mathtt{SAT}_{\mathtt{UP},\mathtt{H},\mathtt{AUTARK}}(.,2)$ *est complet pour les ensembles de clauses de cardinalité binaires.*

L'intérêt des résultats que nous venons d'établir pour Horn-SAT, les clauses simples et les clauses binaires est que l'algorithme n'a pas besoin de savoir que la formule traitée est Horn-renommable ou composée uniquement de clauses simples ou binaires pour atteindre cette complexité. Dans [RAU 95] et section 3.10, il est montré expérimentalement que $\mathtt{SAT}_{\mathtt{UP},\mathtt{BIMO},\mathtt{AUTARK}}$ est en pratique linéaire sur des ensembles de clauses binaires tirés aléatoirement, égalant ainsi les meilleurs algorithmes de reconnaissance [ASP 80, MAN 85, ARV 87, LIN 89, CHA 90, HÉB 94] et de décision [DOW 84, MIN 88, GHA 91] connus pour les clauses de Horn, ainsi que les meilleurs algorithmes de décision connus pour les ensembles de clauses binaires (voir, par exemple, les références [ASP 79, EVE 76, PET 91]). Ceci confirme les résultats de [PET 91], où divers algorithmes pour résoudre 2-SAT sont comparés et où le plus efficace en pratique s'avère être un algorithme énumératif dont la complexité est quadratique.

3.6. *Horn-renommage*

Le problème du Horn-renommage consiste à déterminer si un ensemble de clauses peut être renommé en un ensemble de clauses de Horn. Ce problème a intéressé les chercheurs depuis les premières heures de la déduction automatique car les clauses de Horn y jouent un rôle central. De nombreux articles [MEL 66, LEW 78, ASP 80, MAN 85, LIN 89] ont été publiés avant qu'on ne réussisse à définir un algorithme de complexité linéaire [CHA 90, HÉB 94].

Dire qu'un ensemble de clauses est de Horn, c'est dire que chacune de ses clauses contient au plus un littéral positif. Par conséquent, un ensemble est Horn-renommable si et seulement s'il existe une affectation falsifiant au plus un littéral par clause (il suffit ensuite de renommer les variables ayant reçu la valeur 1). Pour résoudre le problème du Horn-renommage associé à un ensemble de clauses classiques, il suffit donc de transformer chaque clause C de l'ensemble en la clause de cardinalité $(|C|-1,|C|,C)$. D'où le corollaire suivant, qui est une conséquence directe du théorème 6.

Corollaire 8 (Horn-renommage) *Le schéma* χ-$\mathtt{SAT}_{\mathtt{UP,H,AUTARK}}(.,2)$ *est complet pour les ensembles de clauses de cardinalité codant un problème de Horn-renommage.*

Comme on peut s'y attendre, le schéma $\mathtt{SAT}_{\mathtt{UP,BIMO,AUTARK}}$ s'avère linéaire en pratique sur les problèmes de renommage issus d'ensembles de clauses tirés aléatoirement (voir section 3.10).

Dans [HÉB 95], Hébrard propose un algorithme linéaire subtil pour déterminer si un ensemble de clauses est uniquement Horn-renommable (et ainsi, si un ensemble de clauses binaires n'a qu'une seule solution). Avec le schéma $\mathtt{SAT}_{\mathtt{UP,H,AUTARK}}$, il suffit simplement, pour répondre à cette question, de continuer l'exploration après que la première solution a été trouvée (jusqu'à en trouver une seconde ou prouver qu'il n'y en a pas d'autres). Il est facile de vérifier que cela n'augmente pas la complexité de l'algorithme. De plus, dans les expérimentations que nous avons menées, les solutions trouvées ne concernent quasiment jamais toutes les variables. Cela signifie qu'elles codent effectivement plusieurs Horn-renommages.

3.7. *Comment rendre linéaire le schéma* χ-$\mathtt{SAT}_{\mathtt{UP,BIMO,AUTARK}}(.,2)$

Nous avons vu que le schéma χ-$\mathtt{SAT}_{\mathtt{UP,H,AUTARK}}(.,2)$ permet de résoudre les principaux problèmes propositionnels polynomiaux : Horn-SAT, 2-SAT, Horn-renommage, qHorn-SAT (pour ce dernier, voir section 3.8).

Il existe des algorithmes linéaires pour résoudre ces problèmes. Or, même avec une heuristique rapide de type BIMO, le schéma χ-$\mathtt{SAT}_{\mathtt{UP,H,AUTARK}}(.,2)$ est quadratique (en $\mathcal{O}(V \times S)$). Cette complexité est effectivement atteinte si l'algorithme choisit un littéral p_1, effectue une propagation unitaire en $\mathcal{O}(S)$, revient en arrière, effectue une propagation unitaire sur $\neg p_1$ en $\mathcal{O}(1)$, choisit un littéral p_2, etc. Cet exemple illustre l'astuce proposée dans [EVE 76] pour obtenir une complexité linéaire. Elle consiste à faire la propagation unitaire pour p_1 et $\neg p_1$ de manière concurrente (disons en entrelaçant les visites des occurrences positives et négatives de p_1). Si l'une des deux propagations se termine sur un succès, alors l'autre est abandonnée. Ainsi, le cas critique décrit plus haut ne peut pas se produire. Fondamentalement, c'est ce que Hébrard a proposé dans [HÉB 94] (même s'il n'a pas formulé les choses de la même manière).

3.8. *q-Horn Clauses*

La classe des q-Horn clauses a été introduite dans [BOR 90]. Elle généralise à la fois les clauses de Horn et les clauses binaires. Un ensemble de clauses est dit *q-Horn* s'il existe une partition des variables en deux sous-ensembles disjoints H et B tels que :

– aucune clause ne contient plus de deux variables de B ;

– aucune clause ne contient plus d'un littéral positif de H ;

– aucune clause contenant un littéral positif de H ne contient un littéral de B.

Comme pour les clauses de Horn, on dit qu'un ensemble de clauses est q-Horn renommable s'il existe un renommage qui le transforme en un ensemble q-Horn.

Si les ensembles H et B sont connus, il est facile de décider de la satisfiabilité d'un ensemble q-Horn. On affecte la valeur 0 à toutes les variables de H et on résout le problème restant, qui ne contient plus que des clauses binaires. Dans [BOR 94b], un algorithme linéaire est proposé pour reconnaître B et H et dans [BOR 94a] un indice de complexité est proposé qui utilise cet algorithme.

Pour résoudre efficacement une instance q-Horn ϕ, on peut aussi utiliser le schéma $\texttt{SAT}_{\texttt{UP},\texttt{UPLA},\texttt{AUTARK}}$.

Théorème 9 (q-Horn SAT) *Le schéma* $\chi\text{-}\texttt{SAT}_{\texttt{UP},\texttt{UPLA},\texttt{AUTARK}}(.,2)$ *est complet pour les ensembles de clauses q-Horn renommables.*

Soit ϕ un ensemble de clauses q-Horn renommable. On peut supposer sans perte de généralité que ϕ ne contient ni littéral unitaire, ni littéral pur. Soit ϕ^B le sous-ensemble de ϕ construit uniquement sur les variables de B. ϕ^B est un ensemble de clauses binaires et ϕ est satisfiable si et seulement si ϕ^B est satisfiable (les autres clauses sont satisfaites en affectant la valeur 0 à tous les littéraux de H). Or, ϕ^B est insatisfiable si et seulement s'il existe une variable x telle que $\texttt{UP}(\phi^B_x)$ et $\texttt{UP}(\phi^B_{\neg x})$ contiennent tous deux la clause vide (c'est une conséquence du principe d'autark), ce qui est détecté par UPLA. Supposons donc que ϕ soit satisfiable et simplifié. Soit p le littéral choisi par UPLA. Deux cas sont possibles. Si ϕ_p est insatisfiable, l'application de UPLA sur $\texttt{UP}(\phi_p)$ le détecte (en $\mathcal{O}(V \times S)$). Si $\texttt{UP}(\phi_p)$ est satisfiable, on n'a pas besoin de rappeler l'algorithme sur $\phi_{\neg p}$. □

D'où le corollaire suivant, qui est en fait vrai pour toute classe de formules sur laquelle le schéma $\chi\text{-}\texttt{SAT}_{\texttt{UP},\texttt{UPLA},\texttt{AUTARK}}$ est complet.

Corollaire 10 (q-Horn SAT) *q-Horn SAT* $\in \aleph^{\star}_{\texttt{UP},\texttt{AUTARK},2}$.

Compter le nombre de solutions d'un ensemble de clauses de Horn est un problème #P-complet [VAL 79]. Par conséquent, les heuristiques de la catégorie de FFIS, si elles choisissent systématiquement les littéraux de H avant ceux de B, ne peuvent garantir la polynomialité de l'algorithme sur les formules insatisfiables. En d'autres termes, q-Horn SAT $\notin \aleph_{\texttt{UP},\texttt{FFIS},\texttt{AUTARK}}$.

On peut généraliser aux contraintes de cardinalité le principe des qHorn-clauses : l'ensemble de clauses binaires peut être un ensemble de clauses de cardinalité simples.

3.9. *Hiérarchie de Dalal et Etherington*

Pour finir cette étude des problèmes propositionnels polynomiaux, nous compa-

rerons maintenant la hiérarchie $\aleph^{\star}_{UP,AUTARK,(n)}$ définie section 2.4 à celle de Dalal et Etherington [DAL 92c]. Cette dernière, qui généralise celle de Gallo et Scutellà [GAL 88], est consituée de deux hiérarchies entrelacées, $\Delta_{(n)}$ et $\Omega_{(n)}$ définies comme suit :

– la classe Δ_0 contient les formules ϕ telles que $\mathtt{UP}(\phi)$ contient la clause vide ou ne contient pas de clause positive (ne contenant que des littéraux positifs) ou ne contient pas de clause négative ;

– pour tout k, $\phi \in \Omega_k$, si et seulement si ou bien $\phi \in \Delta_k$ ou bien pour tout littéral p, ou bien $\mathtt{UP}(\phi_p) \in \Delta_k$ et $\mathtt{UP}(\phi_{\neg p}) \in \Omega_k$, ou bien $\mathtt{UP}(\phi_p) \subset \phi$ et $\mathtt{UP}(\phi_p) \in \Omega_k$;

– pour tout $k > 0$, $\phi \in \Delta_k$, si et seulement si il existe un littéral p tel que, ou bien $\mathtt{UP}(\phi_p) \in \Omega_{k-1}$ et $\mathtt{UP}(\phi_{\neg p}) \in \Delta_k$, ou bien $\mathtt{UP}(\phi_p) \subset \phi$ et $\mathtt{UP}(\phi_p) \in \Delta_k$.

La classe Δ_0 est trivialement décidable. En effet, après propagation unitaire, il suffit de compter le nombre de littéraux positifs et négatifs par clause. Si l'instance ne contient pas de clause positive (resp. négative), l'affection de la valeur 0 (resp. 1) à toutes les variables satisfait l'instance. Ce test revient à privilégier deux affectations totales particulières (tout à 1 et tout à 0) parmi les 2^V possibles. De ce point de vue, cette classe est parfaitement arbitraire et pourrait être généralisée, par exemple en testant un nombre polynomial d'affectations totales : celles étant à une distance de Hamming inférieure à une constante k d'une certaine affection de référence. De plus, Δ_0 n'est pas close par affectation. En effet, une formule ϕ peut être dans Δ_0, alors que ϕ_p n'y est pas (p étant un littéral). Ce qui constitue un argument supplémentaire pour dénoncer son caractère artificiel.

La construction de la hiérarchie $\Delta_{(n)}$ suit le même principe que celle de la hiérarchie $\aleph^{\star}_{UP,AUTARK,(n)}$: recherche d'un littéral et décrémentation d'une borne sur l'une des branches (et, éventuellement, détection des cas d'autark). Toutefois, ce processus est ici quelque peu brouillé par l'introduction des Ω_i. Ces derniers permettent de gagner un littéral "en profondeur" (il faut deux affectations entre Δ_i et Δ_{i-1}). Cependant, le fait d'exiger que toutes les variables doivent avoir une certaine propriété rend la hiérarchie $\Omega_{(n)}$ très artificielle. Il ne doit d'ailleurs exister que peu de formules appartenant à $\Omega_i \setminus \Delta_{i-1}$.

Quoi qu'il en soit, pour tout entier k, on a $\aleph^{\star}_{UP,AUTARK,k+1} \subset \Delta_k$ (par construction) et on peut construire des formules ϕ telles que $\phi \notin \Delta_{k-1}$ et $\phi \in \aleph^{\star}_{UP,AUTARK,k+1}$. Considérons en effet la famille de formules $\phi_{(n)}$ suivante :

$$\begin{aligned}
\phi_0 &= \{\{x_0,y_0\},\{x_0,\neg y_0\},\{\neg x_0\}\} \\
\phi_{i+1} &= \{\{x_{i+1},y_{i+1}\} \cup C; C \in \phi_i\} \cup \{\{x_{i+1},\neg y_{i+1}\} \cup C; C \in \phi_i\} \\
&\quad \cup \{\{\neg x_{i+1}\} \cup C; C \in \phi_i\}
\end{aligned}$$

Il est facile de vérifier que, pour $k > 2$, on a $\phi_k \in \aleph^{\star}_{UP,AUTARK,k+1}$ et $\phi_k \notin \Delta_{k-1}$.

3.10. *Expérimentations*

Nous avons testé le schéma $\mathtt{SAT}_{\mathtt{UP,BIMO,AUTARK}}$ sur (entre autres) des ensembles de clauses binaires générés aléatoirement suivant le modèle dit de taille fixée [DUB 96]. Dans ce modèle, on se donne le nombre de variables V et le nombre de clauses C. On tire les C clauses indépendamment. Pour chaque clause, on tire deux variables (distinctes) parmi les V, puis on tire une polarité pour chaque variable, les deux signes étant équiprobables. Le générateur de nombres pseudo-aléatoires utilisé est celui préconisé dans [PRE 92]. Notre mise en œuvre du schéma $\mathtt{SAT}_{\mathtt{UP,BIMO,AUTARK}}$ comporte moins de 500 lignes de C, parseur, aide en ligne et affichage des résultats compris.

Pour les raisons évoquées section 3.6, un ensemble de clauses binaires est Horn-renommable si et seulement si il est satisfiable. Les clauses binaires sont les plus difficiles pour le Horn-renommage, parce que ce sont celles qui provoquent le moins de propagations. Démontrer l'efficacité du schéma $\mathtt{SAT}_{\mathtt{UP,BIMO,AUTARK}}$ sur 2-SAT, démontre donc *a fortiori* son efficacité pour le Horn-renommage.

Le tableau 1 présente les résultats obtenus en générant des instances pour $V = 10^6$. Il est clair qu'un algorithme de résolution qui ne serait pas linéaire ne pourrait pas traiter des formules de cette taille en des temps raisonnables. Chaque ligne correspond aux résultats obtenus pour une valeur du rapport C/V. La $1^{ière}$ colonne indique ce dernier. Les $2^{ième}$, $3^{ième}$ et $4^{ième}$ colonnes contiennent respectivement le nombre d'instances considérées (#ins.), le nombre de formules satisfiables (#SAT) parmi celles-ci et la moyenne du nombre de variables distinctes apparaissant effectivement dans l'instance (#var). Pour les faibles valeurs du rapport C/V, ce nombre peut être significativement inférieur à V. Les $5^{ième}$ et $6^{ième}$ colonnes contiennent respectivement la moyenne du nombre d'affectations effectuées (#ass.) et la moyenne du temps de calcul nécessaire pour résoudre l'instance (rés.). Finalement, les $7^{ième}$ et $8^{ième}$ colonnes contiennent respectivement le temps nécessaire pour générer une instance (gén.) et le temps nécessaire pour la lire (lect.)[2]. Tous les temps sont des temps CPU mesurés en secondes.

La proportion de formules satisfiables passe brutalement de 1 à 0 quand le rapport C/V dépasse 1 (comme le montre clairement la courbe dessinée sur la figure 4). C'est le phénomène de transition de phase désormais bien connu (et qui été établi analytiquement pour 2-SAT [GOE 92, CHV 92]).

Le nombre d'affectations est toujours inférieur au nombre de variables effectivement présentes dans l'instance (alors que le principe d'autark n'est quasiment jamais appliqué et que les deux branches d'une alternative sont explorées séquentiellement). De ce point de vue, le schéma $\mathtt{SAT}_{\mathtt{UP,BIMO,AUTARK}}$ a une complexité sous-linéaire. Sur la figure 5, on a tracé la courbe du rapport de (la moyenne de) ces deux nombres (en fonction du rapport C/V) et on a séparé les formules satisfiables et insatisfiables. En procédant ainsi, il apparaît clairement que pour les formules satisfiables, le nombre

2. Ces temps sont relativement imprécis dans la mesure où les opérations mettent en œuvre des appels systèmes dont l'efficacité dépend de la charge de la machine. Ils donnent toutefois une indication sur l'ordre de grandeur des temps moyens.

C/V	#ins.	#SAT	#var.	#ass.	rés.	gén.	lect.
0.500	20	20	632185	631765	40.92	104.20	73.00
0.550	20	20	667167	666710	43.93	114.50	72.70
0.600	20	20	698811	698365	46.86	124.10	72.60
0.650	20	20	727492	727080	49.87	134.30	74.60
0.700	20	20	753405	753056	52.70	145.60	73.70
0.750	20	20	776827	776430	55.66	155.80	77.80
0.800	100	100	798167	797691	57.82	166.90	82.00
0.850	100	100	817371	816913	60.39	176.90	77.90
0.900	100	100	834746	834317	63.02	187.10	86.30
0.950	100	100	850480	850141	66.00	196.30	81.80
0.980	200	198	859251	855474	67.24	205.30	83.40
0.985	200	197	860654	855137	67.51	206.40	87.60
0.990	200	191	862041	846281	67.16	205.40	81.80
0.995	200	184	863415	836270	66.56	209.10	78.30
1.000	200	176	864712	822320	65.53	208.70	79.20
1.005	200	151	866122	784001	63.09	208.50	90.60
1.010	200	120	867572	734764	59.63	212.40	82.40
1.015	200	78	868770	661989	54.21	210.10	83.50
1.020	200	40	870187	596530	49.11	213.10	79.50
1.025	200	10	871369	547701	45.76	214.10	84.60
1.030	100	0	872756	510009	42.04	244.80	58.10
1.040	100	0	875288	494338	40.45	217.20	85.40
1.050	100	0	877591	483295	38.84	217.20	84.80
1.100	100	0	889256	458629	36.76	228.60	81.40
1.150	100	0	899800	442120	35.52	239.90	83.60
1.200	100	0	909332	428541	34.60	248.40	93.90
1.250	100	0	917965	414597	34.12	263.20	86.10
1.300	20	0	925729	401163	33.09	269.40	96.60
1.350	20	0	932811	388692	32.44	284.40	92.30
1.400	20	0	939188	375321	31.52	289.40	106.90
1.450	20	0	944955	363195	30.85	305.90	102.40
1.500	20	0	950214	349761	29.89	311.10	91.20
1.550	20	0	954936	337879	29.11	324.90	94.20
1.600	20	0	959250	325500	28.23	331.70	99.50
1.650	20	0	963129	313205	27.32	341.90	104.20
1.700	20	0	966630	300880	26.48	354.90	102.50
1.750	20	0	969807	289682	25.74	364.40	92.70
1.800	20	0	972702	278018	24.93	427.20	95.50

Tableau 3. *Résultats pour des instances 2-SAT avec* 10^6 *variables*

			temps ensemble		temps SAT		temps UNSAT	
C/V	#ins.	#SAT	moy.	σ	moy.	σ	moy.	σ
0.500	20	20	40.92	0.20	40.92	0.20	_	_
0.550	20	20	43.93	0.19	43.93	0.19	_	_
0.600	20	20	46.86	0.16	46.86	0.16	_	_
0.650	20	20	49.87	0.14	49.87	0.14	_	_
0.700	20	20	52.70	0.13	52.70	0.13	_	_
0.750	20	20	55.66	0.16	55.66	0.16	_	_
0.800	100	100	57.82	0.32	57.82	0.32	_	_
0.850	100	100	60.39	0.27	60.39	0.27	_	_
0.900	100	100	63.02	0.25	63.02	0.25	_	_
0.950	100	100	66.00	0.26	66.00	0.26	_	_
0.980	200	198	67.24	2.85	67.53	0.17	39.10	2.70
0.985	200	197	67.51	3.59	67.95	0.25	38.60	2.57
0.990	200	191	67.16	5.70	68.39	0.28	41.07	2.92
0.995	200	184	66.56	7.17	68.64	0.34	42.65	4.46
1.000	200	176	65.53	8.62	68.79	0.68	43.74	4.72
1.005	200	151	63.09	11.43	69.40	0.56	43.65	5.66
1.010	200	120	59.63	13.02	69.58	0.66	44.71	7.22
1.015	200	78	54.21	13.66	70.28	0.81	43.93	5.88
1.020	200	40	49.11	12.51	69.90	0.30	43.91	7.77
1.025	200	10	45.76	9.33	72.23	3.27	44.37	7.23
1.030	100	0	42.04	4.47	_	_	42.04	4.47
1.040	100	0	40.45	4.47	_	_	40.45	4.47
1.050	100	0	38.84	2.94	_	_	38.84	2.94
1.100	100	0	36.76	1.14	_	_	36.76	1.14
1.150	100	0	35.52	0.79	_	_	35.52	0.79
1.200	100	0	34.60	0.71	_	_	34.60	0.71
1.250	100	0	34.12	0.65	_	_	34.12	0.65
1.300	20	0	33.09	0.37	_	_	33.09	0.37
1.350	20	0	32.44	0.38	_	_	32.44	0.38
1.400	20	0	31.52	0.22	_	_	31.52	0.22
1.450	20	0	30.85	0.29	_	_	30.85	0.29
1.500	20	0	29.89	0.32	_	_	29.89	0.32
1.550	20	0	29.11	0.32	_	_	29.11	0.32
1.600	20	0	28.23	0.24	_	_	28.23	0.24
1.650	20	0	27.32	0.22	_	_	27.32	0.22
1.700	20	0	26.48	0.26	_	_	26.48	0.26
1.750	20	0	25.74	0.21	_	_	25.74	0.21
1.800	20	0	24.93	0.23	_	_	24.93	0.23

Tableau 3. *Moyennes et écarts types des temps de calculs.*

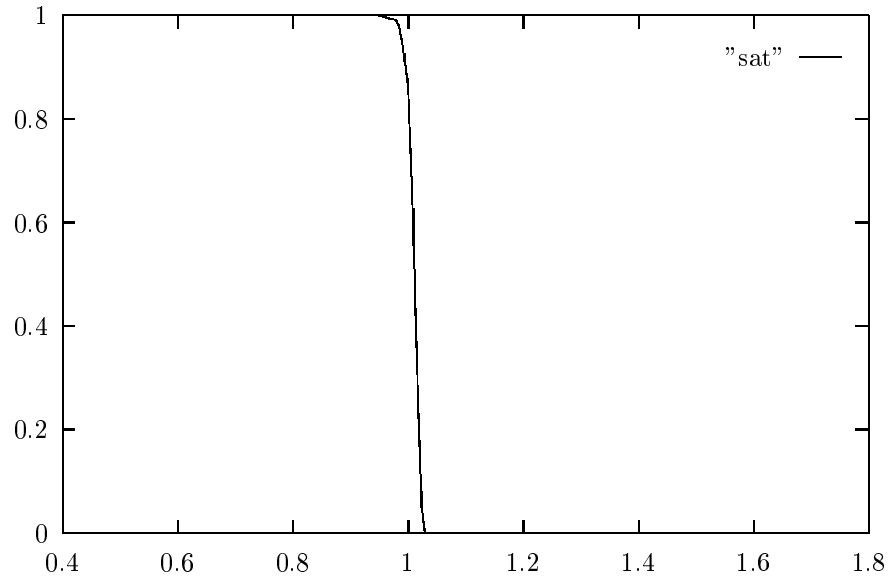


Figure 4. *Proportion de formules satisfiables en fonction du rapport C/V.*

d'affectations est toujours égal, à peu de choses près, au nombre de variables. Pour les formules insatisfiables, qui sont beaucoup plus faciles, ce rapport passe par un pic, puis décroît régulièrement lorsque le rapport C/V augmente. Cela peut s'expliquer de la façon suivante : lorsque l'instance est satisfiable, l'algorithme ne remet quasiment pas en cause les choix effectués et trouve une solution qui affecte presque toutes les variables (mais en pratique jamais toutes les variables). Pour les formules insatisfiables, plus le rapport C/V augmente, plus vite apparaissent les contradictions, d'où la diminution du nombre d'affectations.

Les temps de calcul ne dépassent jamais 75 secondes, pour traiter des formules avec près d'un million de variables et plus d'un million de clauses ! Ce résultat justifie à lui seul l'ensemble de notre travail. En effet, il rend d'aucune utilité pratique les nombeux algorithmes spécialisés proposés pour Horn-SAT, 2-SAT, la Horn-renommabilité et les problèmes dérivés (unique Horn-SAT, unique 2-SAT, etc.)[3]. Tout cela se fait avec une incroyable efficacité grâce à un seul et même algorithme : notre bonne vieille procédure de Davis et Putnam. Sur le tableau 1, on voit d'ailleurs clairement que le temps nécessaire pour résoudre une instance est toujours très inférieur au temps nécessaire pour la générer et au temps nécessaire pour la lire (alors que l'analyseur ne fait que remplir la matrice creuse et que l'accès aux variables se fait en temps constant).

3. Bien entendu, nous ne portons pas là un jugement sur l'intérêt théorique et historique de ces algorithmes.

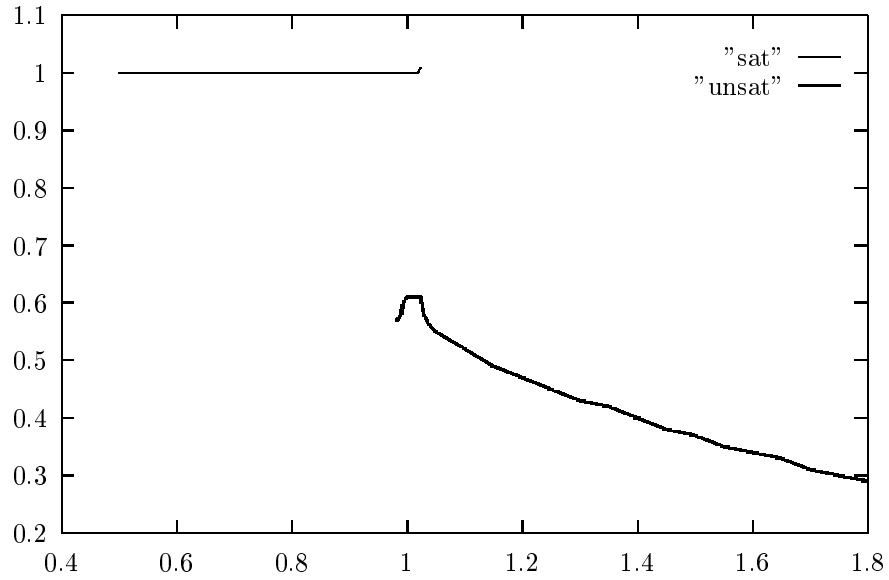


Figure 5. *Rapport du nombre d'affectations effectuées et du nombre de variables effectivement présentes dans l'instance en fonction du rapport C/V.*

Non seulement le schéma $\texttt{SAT}_{\texttt{UP,BIMO,AUTRAK}}$ est efficace en moyenne, mais il est aussi très robuste. Le tableau 2 donne les moyennes et les écarts types des temps de résolution pour toutes les formules, uniquement les formules satisfiables et uniquement les formules insatisfiables. On voit que les écarts types sont très faibles. De plus, un examen rapide des résultats montre que la complexité en temps et la complexité en nombre d'affectations sont très étroitement corrélées.

Nous avons effectué de nombreuses autres expérimentations qui viennent toutes renforcer celles présentées ici. La conclusion est claire : en pratique, lorsqu'une instance est facile, elle est quasiment toujours facile pour le schéma $\texttt{SAT}_{\texttt{UP,H,AUTARK}}$.

4. Problèmes de satisfaction de contraintes

Le formalisme des CSPs (pour Problèmes de Satisfaction de Contraintes) étend les formules CNF en considérant des variables prenant leurs valeurs dans des domaines finis et en permettant des contraintes plus générales que de simples clauses. Dans cette section, nous étendons les résultats obtenus dans les sections précédentes à certaines classes polynomiales des CSPs. Notre objectif est de montrer, comme nous l'avons fait pour les problèmes purement propositionnels, que le schéma $\texttt{SAT}_{\texttt{S,H,E}}$ est efficace sur les traductions en formules CNF des problèmes appartenant à ces classes.

4.1. *Rappels sur les CSP*

Un CSP est un quadruplet $(\mathcal{X},\mathcal{D},\mathcal{C},\mathcal{R})$ défini par :

– un ensemble $\mathcal{X} = \{X_1, \ldots, X_n\}$ de variables ;

– un ensemble $\mathcal{D} = \{D_1, \ldots, D_n\}$ de domaines, c'est-à-dire d'ensembles finis de valeurs. À chaque variable X_i est associé son domaine D_i ;

– un ensemble $\mathcal{C} = \{C_1, \ldots, C_m\}$ de contraintes. Chaque contrainte étant définie par un sous-ensemble des variables $C_i = \{X_{i1}, \ldots, X_{ik_i}\}$;

– un ensemble de relations $\mathcal{R} = \{R_1, \ldots, R_m\}$. Chaque relation R_i étant un sous-ensemble du produit cartésien des domaines associés aux variables de la contrainte C_i ($R_i \subseteq D_{i1} \times \ldots \times D_{ik_i}$).

Lorsque toutes les contraintes portent sur des couples de variables, on parle de CSP binaire. On appelle structure ou hypergraphe sous-jacent d'un CSP $(\mathcal{X},\mathcal{D},\mathcal{C},\mathcal{R})$ l'hypergraphe $(\mathcal{X},\mathcal{C})$. Si le CSP est binaire, la structure est un graphe. Une solution d'un CSP est une affectation des variables satisfaisant simultanément l'ensemble des contraintes. Une affectation satisfait la contrainte C_i si sa projection sur les variables de C_i appartient à R_i. Résoudre un CSP, c'est déterminer s'il admet au moins une solution, ce qui est bien entendu un problème NP-complet.

En général, on définit la taille d'un CSP binaire par n^2d^2, où n est comme indiqué ci-dessus le nombre de variables et d est la taille du domaine maximum. Dans la suite de cette section, nous garderons la même signification à n et d.

Une partie importante de la littérature sur les CSP porte sur la notion de consistance partielle. L'idée est d'éliminer des relations d'un CSP les affectations partielles des variables ne pouvant pas faire partie d'une solution. Cette notion sert donc principalement à filtrer les problèmes : en supprimant ces affectations partielles inconsistantes, on espère faciliter la résolution du CSP. Freuder l'a formalisé comme suit [FRE 78].

Une affectation partielle de k variables $Y_1, \ldots, Y_k$ d'un CSP est consistante si sa projection sur chaque contrainte C_i du CSP portant sur un sous-ensemble des variables $Y_1, \ldots, Y_k$ appartient à la relation R_i associée à C_i.

Un CSP est k-consistant si toute affectation partielle consistante de $k-1$ variables peut être étendue de manière consistante à n'importe quelle $k^{ième}$ variable. Il est fortement k-consistant s'il est 1-,2-,...,k-consistant.

La 1-consistance revenant à vérifier qu'aucun domaine n'est vide, on peut toujours la supposer vérifiée dans la donnée même du problème. On a donc équivalence entre 2-consistance forte et 2-consistance. Dans le cas des CSP binaires, la 2-consistance est aussi appelée consistance d'arc et la 3-consistance consistance de chemin [MAC 77]. La consistance d'arc peut être réalisée en temps linéaire dans la taille du CSP [MOH 86]. Depuis ce dernier article, un certain nombre d'améliorations ont été proposées pour les consistances d'arc et de chemin (voir par exemple [HAN 88, BES 94, BES 99]).

Une autre notion importante, qui donne des informations topologiques sur le CSP, est celle de largeur du graphe sous-jacent (cette notion s'applique principalement aux CSP binaires). Étant donné un ordre sur les sommets, la largeur d'un sommet est le nombre de ses voisins qui le précèdent dans l'ordre. La largeur d'un ordre est le maximum des largeurs de tous les sommets. Enfin, la largeur du CSP est le minimum des largeurs des ordres possibles. Un CSP de largeur 1 est dit arborescent.

Freuder a montré que tout CSP de largeur k vérifiant la $(k+1)$-forte consistance est consistant [FRE 82]. Les meilleurs algorithmes réalisant la k-consistance sont en $\mathcal{O}(n^k d^k)$ [COO 89]. Malgré cela, le résultat de Freuder est d'assez peu d'utilité pratique dans la mesure où, pour obtenir la k-consistance, on est amené à rajouter des contraintes et donc, en général, à augmenter la largeur du graphe sous-jacent. Les CSP arborescents forment toutefois une exception intéressante à cette règle. En effet, réaliser la 2-forte consistance d'un CSP n'en modifie pas la largeur.

4.2. *Codage d'un CSP par une instance SAT*

Deux traductions du formalisme CSP en formules propositionnelles ont été proposées. Une par de Kleer [KLE 89], l'autre par Dalal [DAL 92b]. Dans les deux cas, le principe est de créer une variable propositionnelle $X : v$ pour chaque couple variable/valeur (X,v) ($v \in D_X$). On a ensuite deux séries de clauses : la première pour coder l'appartenance des variables CSP à leur domaine, la seconde pour coder les contraintes.

La première série de clauses est identique dans les deux codages. Le fait qu'une variable X_i prend valeur dans le domaine $D_X = \{v_1, \ldots, v_p\}$, est codé par la clause $X{:}\, v_1 \vee \ldots \vee X{:}\, v_p$. On code également qu'une variable ne peut prendre qu'une valeur à la fois en ajoutant, pour chaque couple de valeurs distinctes (v,w), la clause $\neg X{:}\, v \vee \neg X{:}\, w$. Ces clauses binaires ne sont pas strictement nécessaires du point de vue de la satisfiabilité, aussi les omet-on la plupart du temps. On peut aussi utiliser la contrainte de cardinalité $(1,1,[X{:}\, v_1, \ldots, X{:}\, v_k])$ pour coder simultanément l'appartenance d'une variable à son domaine et l'unicité de sa valeur.

Dans les deux codages, chaque contrainte $C(X_1, \ldots, X_k)$ de relation R est ensuite considérée séparément. Dans le codage de de Kleer, pour chaque k-uplet $(v_1, \ldots, v_k)$ n'appartenant pas à R, on crée la clause suivante (qui interdit le k-uplet correspondant) :

$$\neg X_1{:}\, v_1 \vee \ldots \vee \neg X_k{:}\, v_k$$

L'idée de Dalal est de coder toutes les déductions que permet la contrainte. Pour chaque variable X_i, on regroupe les k-uplets de R suivant la valeur qu'ils donnent aux autres variables. Pour chaque groupe de k-uplets $(v_1, \ldots, v_{i-1}, v_i^1, v_{i+1}, \ldots, v_k)$, …, $(v_1, \ldots, v_{i-1}, v_i^p, v_{i+1}, \ldots, v_k)$, on écrit la clause suivante :

$$\bigwedge_{j \neq i} X_j{:}\, v_j \;\Rightarrow \bigvee_{q=1,\ldots,p} X_i{:}\, v_i^q$$

Il est bien sûr inutile d'ajouter cette contrainte si le groupe contient un k-uplet pour chaque valeur du domaine de X_i. De plus, on réalise la consistance de domaine : si une valeur v du domaine de X n'apparaît dans aucun k-uplet concernant X, on rajoute la clause $\neg X{:}\, v$.

Dans les deux cas, le codage d'un CSP binaire est en $\mathcal{O}(n^2d^2)$, c'est-à-dire linéaire en la taille du CSP.

4.3. *Adaptation du schéma* χ-$\mathtt{SAT}_{\mathtt{UP},\mathtt{H},\mathtt{AUTARK}}$ *au traitement des CSP*

Nous appelons affectation partielle des variables du CSP l'opération consistant à affecter une valeur à certaines variables et à réduire le domaine d'autres variables. Toute affectation partielle peut être assimilée à un ensemble de littéraux. Cet ensemble contient des littéraux positifs $X{:}\, v$ pour les variables X ayant reçu la valeur v et des littéraux négatifs $\neg Y{:}\, w$ pour les variables Y dont la valeur w a été ôtée du domaine. Nous dirons d'une affectation partielle L qu'elle est saturée si plus aucune des trois règles suivantes ne peut s'appliquer.

- S1: S'il existe une variable X et deux valeurs v et w de D_X telles que $X{:}\, v \in L$ et $\neg X{:}\, w \notin L$, alors ajouter $\neg X{:}\, w$ à L.
- S2: S'il existe une variable X et une valeur v de D_X telles que $X{:}\, v \notin L$ et toutes les autres valeurs w de D_X sont telles que $\neg X{:}\, w \notin L$, alors ajouter $X{:}\, v$ à L.
- S3: S'il existe une contrainte $C(X,Y)$ et une valeur v de D_X telles que $\neg X{:}\, v \notin L$ et tous les couples (v,w) de C sont tels que $\neg Y{:}\, w \in L$, alors ajouter $\neg X{:}\, v$ à L.

La saturation du CSP pour les règles S1-3 réalise la consistance d'arc.

La propagation unitaire `UP` simule, sur les deux traductions et grâce aux clauses de domaines, les règles S1 et S2.

`UP` simule, sur le codage à la Dalal, la règle S3. Soit, en effet, une contrainte $C(X,Y)$ et soient (v,w_1), $\ldots (v,w_p)$ les couples de C dont v est le membre droit. Alors, le codage contient la clause $X{:}\, v \Rightarrow Y{:}\, w_1 \vee \ldots \vee Y{:}\, w_p$. Il s'ensuit que si tous les $Y{:}\, w_i$ sont falsifiés, le littéral $\neg X{:}\, v$ est produit par `UP`.

`UP` ne simule pas la règle S3 sur le codage à la de Kleer. Considérons, par exemple, la contrainte $C(X,Y)$, avec $D_X = \{0,1\}$, $D_Y = \{a,b,c\}$ et $R = \{(0,a),(0,b),(1,c)\}$. Les clauses codant C sont les suivantes :

$$\neg X{:}\, 0 \vee \neg Y{:}\, c,\ \neg X{:}\, 1 \vee \neg Y{:}\, a,\ \neg X{:}\, 1 \vee \neg Y{:}\, b$$

Si on falsifie la variable $Y{:}\, c$, ces clauses ne permettent pas à `UP` de produire le littéral $\neg X{:}\, 1$. En revanche, si l'on essaye de satisfaire par la suite $X{:}\, 1$, `UP` produit la clause vide.

D'où les lemmes suivants, qui caractérisent les effets de `UP` sur les codages des CSP.

Lemme 11 (CSP, codage de Dalal et UP) *Soient $\mathcal{P}$ un CSP et L une affectation partielle de variables de $\mathcal{P}$. Soit Δ le codage à la Dalal de $\mathcal{P}$. Alors, les affectations partielles obtenues respectivement par saturation de $\mathcal{P}_L$ pour les règles S1, S2 et S3 et par application de* UP *sur Δ_L sont identiques.*

Lemme 12 (CSP, codage de de Kleer et UP) *Soient $\mathcal{P}$ un CSP et L une affectation partielle de variables de $\mathcal{P}$. Soient Γ les codages à la de Kleer de $\mathcal{P}$. Soient L_S et L_K les affectations partielles obtenues respectivement par saturation de $\mathcal{P}_L$ pour les règles S1, S2 et S3 et par application de* UP *sur Γ_L. Alors, on est dans un des deux cas suivants :*

– $L_S = L_K$,

– $L_K \subset L_S$ et il existe une variable X et une valeur v telles que $\neg X{:}\, v \in L_S \setminus L$ et l'application de UP *sur $\Gamma_{L \cup \{X_v\}}$ produit la clause vide.*

Une des conséquences du lemme 12 est que sur le codage à la de Kleer la consistance d'arc est réalisée sans retour arrière par appels successifs à une heuristique de type UPLA.

Considérons une affectation partielle saturée L qui donne une valeur à un certain nombre de variables et ne réduit pas le domaine des autres variables. Si L est consistante, $\mathcal{P}$ est satisfiable si et seulement si $\mathcal{P}_L$ l'est.

Dans le codage à la de Kleer, l'affectation L satisfait les clauses de domaine de variables affectées. Par hypothèse, elle satisfait aussi toutes les clauses de la forme $\neg X{:}\, v \vee \neg Y{:}\, w$ touchées par l'affectation. On est donc dans un cas d'autark.

Dans le codage à la Dalal, l'affectation L satisfait les clauses de domaine de variables affectées. Elle satisfait aussi les clauses $X : v \Rightarrow \bigvee_p Y : w_p$ si une valeur différente de v a été affectée à X ou si une valeur (quelconque) a été affectée à Y. En revanche, elle raccourcit ces même clauses si la valeur v a été affectée à X et aucune valeur n'a été affectée à Y. En fait, comme par hypothèse L ne réduit aucun domaine, la disjonction $\bigvee_p Y : w_p$ est subsumée par la clause de domaine de Y. On est donc aussi dans un cas d'autark si la procédure de détection d'autark est capable de repérer ce type de subsomptions. Cela peut être fait en temps constant, au prix toutefois d'une légère spécialisation des matrices creuses : il suffit de vérifier que le nombre de littéraux positifs dans les clauses $X{:}\, v \Rightarrow \bigvee_p Y{:}\, w_p$ est supérieur ou égal au nombre de littéraux de la clause de domaine de Y.

Le codage en formules propositionnelles de CSP introduit une asymétrie fondamentale entre les littéraux positifs et les littéraux négatifs. Dans une solution, on a plusieurs littéraux négatifs pour chaque littéral positif (si les domaines ne sont pas binaires). D'où l'idée de spécialiser légèrement le schéma χ-$\mathtt{SAT}_{\mathtt{S,H,E}}(.,k_{max})$ de façon à ce que k_{max} soit systématiquement décrémenté sur la branche correspondant au littéral positif (et donc laissé tel quel sur l'autre branche). Dans la suite de cette section, nous considérerons le schéma χ-$\mathtt{SAT}_{\mathtt{S,H,E}}(.,k_{max})$ ainsi spécialisé.

4.4. *CSP Arborescents*

Comme il a été dit précédemment, il existe des algorithmes linéaires pour décider les CSP binaires arborescents. On a le théorème suivant.

Théorème 13 (CSP arborescents) *Le schéma* χ-$\mathtt{SAT}_{\mathtt{UP},\mathtt{H},\mathtt{AUTARK}}(.,1)$ *est complet pour les formules codant à la Dalal des CSP arborescents. Il est complet pour* `H=UPLA` *pour les formules codant à la de Kleer ces CSP.*

Pour le codage à la Dalal, la démonstration est essentiellement la même que pour les clauses de Horn (théorème 5) : soit $X : v$ la variable choisie par l'heuristique `H`. Si l'affectation de la valeur choisie à $X : v$ produit un CSP inconsistant, cela sera détecté par propagation unitaire. Sinon, on n'aura pas besoin de remettre en question cette affectation. Pour le codage à la de Kleer, la situation est un peu plus compliquée, puisque ce dernier ne réalise pas la consistance d'arc. En revanche, on remarque que s'il existe une variable X, une valeur v et une contrainte C telles que l'affectation de v à X falsifie C, alors le codage de C va détecter cette inconsistance (d'après le lemme 12). Il s'ensuit qu'une heuristique de type `UPLA` réalise l'arc consistance. La démonstration est donc assez analogue à celle du théorème 9 portant sur les qHorn clauses. □

4.5. *CSP Zéro-Un-Tous*

Dans [COO 94], Cooper et col. ont défini une restriction des CSP binaires, portant sur la forme des contraintes, pour laquelle on dispose d'un algorithme de résolution polynomial et, ce qui est plus intéressant encore, ont montré que cette restriction est en quelque sorte la plus faible possible. Cooper et col. présentent cette restriction comme une généralisation de 2-SAT.

Un CSP Zéro-Un-Tous contient trois types de contraintes :

– des contraintes *complètes* $C(X,Y)$, dont les relations sont de la forme $A \times B$ pour $A \subseteq D_X$ et $B \subseteq D_Y$;

– des contraintes *permutations* $C(X,Y)$, dont les relations sont de la forme $\{(v,\pi(v))|v \in A\}$ pour $A \subseteq D_X$, $B \subseteq D_Y$ et une bijection $\pi : A \rightarrow B$;

– des contraintes *en double éventail* $C(X,Y)$, dont les relations R sont telles qu'il existe $v \in A \subseteq D_X$ et $w \in B \subseteq D_Y$ tels que $R = (v \times B) \cup (A \times w)$.

Ces trois types de contraintes sont schématisés sur la figure 6.

Soit $\mathcal{P}$ un CSP ZUT et soient Δ et Γ les codages à la Dalal et à la de Kleer de $\mathcal{P}$. On suppose que $\mathcal{P}$ est simplifié par consistance d'arc. On a vu dans la section 4.2 que la propagation unitaire simule cette simplification sur le codage à la Dalal et que les heuristiques `UPLA` font la même chose sur le codage à la de Kleer. Cette supposition

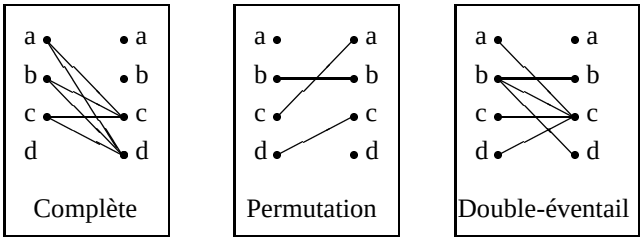


Figure 6. *Les trois types de contraintes des CSP ZUT*

peut donc être faite sans perte de généralité (par rapport à la polynomialité du processus). Soit $X_i{:}\ v$ une variable apparaissant dans Δ (et donc dans Γ) et soit $C(X_i,X_j)$ une contrainte de $\mathcal{P}$. Alors, on se trouve dans un des deux cas suivants :

– l'affectation de v à X_i est compatible avec toutes les affectations de valeur à X_j. Dans ce cas, les clauses de Δ de la forme $X_i{:}\ v \Rightarrow \bigvee_p X_j{:}\ w_p$ sont subsumées par la clause de domaine de X_i et Γ ne contient pas de clause (non satisfaite) de la forme $\neg X_i{:}\ v \vee \neg X_j{:}\ w$. Donc si toutes les contraintes portant sur X_i et v sont de cette forme, les formules $\mathtt{UP}(\Delta_{X_i v})$ et $\mathtt{UP}(\Gamma_{X_i v})$ ne contiennent aucune clause raccourcie ;

– l'affectation de v à X_i n'est compatible qu'avec l'affectation de la valeur w de X_j. Dans ce cas, UP appliquée sur $\Delta_{X_i v}$ et sur $\Gamma_{X_i v}$ produit $X_j{:}\ w$.

En d'autres termes, ou bien la propagation unitaire produit une contradiction, ou bien elle ne produit pas de clause raccourcie : un cas autark (voir plus haut). D'où le théorème suivant.

Théorème 14 (CSP ZUT) *Le schéma* χ-$\mathtt{SAT}_{\mathtt{UP,H,AUTARK}}(.,1)$ *est complet sur les formules codant à la Dalal des CSP Zéro-Un-Tous. Il est complet pour* H=UPLA *sur les formules codant à la de Kleer ces CSP.*

5. Conclusion

Notre travail a consisté à analyser ce qu'il est possible de faire dans le cadre des principales restrictions polynomiales du problème SAT et de deux restrictions polynomiales des CSP en utilisant un schéma énumératif très général. Notre contribution consiste à :

– une présentation des principales classes polynomiales du problème SAT (Horn-SAT, 2SAT, qHorn-SAT, hiérarchie de Gallo et Scutella, hiérarchie de Dalal et Etherington), et cela était nécessaire tant la littérature est abondante et éparse. À l'exception des clauses imbriquées dont il sera question plus loin, nous avons traités ici toutes les classes polynomiales dont nous avions connaissance ;

– l'extension des notions de littéral unitaire et pur, de celle d'autark et des résultats de codage des clauses aux formules de cardinalité ;

– l'utilisation du nombre de Strahler comme mesure de la complexité des démonstrations effectuées par les méthodes énumératives. À notre connaissance, c'est la première fois que cette notion est utilisée dans le cadre de la démonstration automatique. Nous sommes convaincus qu'elle est très puissante et qu'elle servira dans l'avenir ;

– une vision unificatrice de ces classes susmentionnées par le biais d'un algorithme énumératif à nombre de Strahler borné. Cet algorithme s'appuie sur trois principes clefs : la propagation unitaire (filtrage), le principe d'autark (retour arrière intelligent) et la borne du nombre de Strahler de l'arbre d'appels (contrôle de la taille de l'arbre) ;

– l'introduction d'une hiérarchie de formules décidables en temps polynomial. Cette hiérarchie est à la fois raisonnable et robuste ;

– une discussion sur les pontages SAT/CSP ;

– l'application des idées développées sur SAT aux traductions propositionnelles de certaines classes de CSP, afin d'obtenir des pontages algorithmiques effectifs.

Notre travail montre donc qu'un seul algorithme tenant en quelques cinq cent lignes de C est capable de capturer l'essentiel de ce qui fait la polynomialité du problème SAT et dans une moindre mesure des CSP, tout en restant complet. Qui plus est, comme le montrent les résultats expérimentaux, cet algorithme est remarquablement efficace sur ce type de problèmes faciles. Le fait qu'il soit possible de traiter en quelques secondes des ensembles de clauses binaires à un million de variables et autant de clauses ne laisse aucun doute là-dessus.

Ce résultat peut paraître assez négatif, dans la mesure où il semble remettre en cause un certain nombre de travaux effectués sur le sujet. Il ne faut cependant pas se tromper d'optique. Il ne vaut que si l'on adopte le point de vue pratique, c'est-à-dire celui de la réalisation effective de démonstrateurs généralistes. Autrement dit, les travaux sur les restrictions polynomiales conservent tout leur intérêt théorique et historique. Mais pas plus. Car il faut bien appliquer, en science comme ailleurs, le bon vieux rasoir d'Ockham pour départager ce qui est utile de ce qui ne l'est pas.

Notre résultat donne en tout cas un début d'explication à la primauté des méthodes énumératives sur les autres. Si elles se comportent aussi bien en général, c'est en partie parce qu'elles gèrent bien les cas faciles. En outre, ce résultat a une implication pratique : il semble inutile d'inclure des algorithmes dédiés aux cas polynomiaux dans les démonstrateurs généraux.

Quoi qu'il en soit, certaines classes résistent très sérieusement au traitement par une méthode à base d'affectations, notamment les clauses imbriquées de Knuth (voir [KNU 90]). Même s'il semble difficile de construire des instances appartenant à cette classe, ce problème est génant car il semeble indiquer qu'il y a d'autres principes sous-jacents à la polynomialité. Une étude reste à faire à ce sujet.

De même, il serait intéressant d'explorer plus avant les classes polynomiales des CSP (voir par exemple [COO 95]).

Remerciements

Cet article a connu différentes versions depuis la première, écrite en 1996. Les relecteurs, à une exception près, nous ont permis, par leurs critiques pertinentes, d'en améliorer significativement le fond et la forme. Nous tenons à les remercier.

La plupart des résultats présentés ici étaient acquis dès 1996. Il y a une réelle difficulté à faire admettre que certains algorithmes/théorèmes/résultats ne sont pas utiles. Et pourtant, comme le souligne fort justement B. Latour [LAT 88, LAT 89], la science devrait être symétrique, c'est-à-dire faire autant de place à ce qui « marche », qu'à ce qui « ne marche pas ».

6. Bibliographie

[ARV 87] ARVIND V., BISWAS S., « An $\mathcal{O}(n^2)$ algorithm for the satisfiability problem of a subset of propositional sentences in CNF that includes Horn sentences », *Information Processing Letters*, vol. 24, 1987, p. 67–69.

[ASP 79] ASPVALL B., PLASS M., TARJAN R., « A Linear Time Algorithm for Testing the Truth of Certain Quantified Boolean Formulae », *Information Processing Letters*, vol. 8, n° 3, 1979, p. 121–123, Hermes.

[ASP 80] ASPVALL B., « Recognizing Disguised NR(1) Instances of the Satisfiability Problem », *Journal of Algorithms*, vol. 1, 1980, p. 97-103.

[BES 94] BESSIÈRE C., « Arc Consistency and Arc Consistency again », *Artificial Intelligence*, vol. 65, 1994, p. 179–190.

[BES 96] BESSIÈRE C., RÉGIN J.-C., « MAC and combined heuristics: two reasons to forsake FC and CBJ on hard problems », *Proceedings of Conference on Constraint Programming, CP'96*, 1996, p. 61–75.

[BES 99] BESSIÈRE C., FREUDER E., RÉGIN J., « Using Constraint Metaknowledge to Reduce Arc Consistency Computation », *Artificial Intelligence*, vol. 107, 1999, p. 125–148, Elsevier.

[BOR 90] BOROS E., CRAMA Y., HAMMER P., « Polynomial-time inference of all implications for Horn and related formulae », *Annals of Mathematics and Artificial Intelligence*, vol. 1, 1990, p. 21–32.

[BOR 94a] BOROS E., CRAMA Y., HAMMER P., SAKS M., « A complexity index for satisfiability problems », *SIAM Journ. Comp.*, vol. 23, 1994, p. 45–49.

[BOR 94b] BOROS E., HAMMER P., SUN X., « Recognition of q-Horn formulae in linear time », *Discrete Applied Mathematics*, vol. 55, 1994, p. 1–13.

[CHA 90] CHANDRU V., COULARD C., HAMMER P., MONTANEZ M., SUN X., « On renamable Horn and generalized Horn functions », *Annals of Mathematics and Artificial Intelligence*, vol. 1, J.C. Baltzer AG, Scientific Publishing Company, Basel Switzerland, 1990.

[CHV 92] CHVÁTAL V., REED B., « Miks gets some (the odds are on his side) », *Proceedings of the 33rd IEEE Symp. on Foundations of Computer Science*, 1992, p. 620–627.

[COO 71] COOK S., « The Complexity of Theorem Proving Procedures », *Proceedings of the 3rd Ann. Symp. on Theory of Computing, ACM*, 1971, p. 151–158.

[COO 89] COOPER M., « An optimal k-consistency Algorithm », *Artificial Intelligence*, vol. 41, 1989, p. 89–95.

[COO 94] COOPER M., COHEN D., JEAVONS P., « Characterizing Tractable Constraints », *Artificial Intelligence*, vol. 65, 1994, p. 347–361, Elsevier Science Publishers.

[COO 95] COOPER M., JEAVONS P., « Tractable Constraints on Ordered Domains », *Artificial Intelligence*, vol. 79, 1995, p. 327–339, Elsevier Science Publishers.

[CRA 96] CRAWFORD J., AUTON L., « Experimental Results on the Crossover Point in Random 3SAT », *Artificial Intelligence, Special issue on transition phases in problem states*, vol. 81, 1996, p. 31–57.

[DAL 92a] DALAL M., « Efficient Propositional Constraint Propagation », *Proceedings of the* 10^{th} *National Conference on Artificial Intelligence, AAAI'92*, 1992, p. 409–414, San Jose, California.

[DAL 92b] DALAL M., « Tractable Deduction in Knowledge Representation Systems », *Proceedings of the* 3^{th} *International Conference on Principles of Knowledge Representation and Reasoning, KR'92*, 1992, p. 393–402, Boston MA.

[DAL 92c] DALAL M., ETHERINGTON D., « A hierarchy of tractable satisfiability problems », *Information Processing Letters*, vol. 44, 1992, p. 173–180.

[DAV 62] DAVIS M., LOGEMANN G., LOVELAND D., « A Machine Program for Theorem Proving », *CACM*, vol. 5, 1962, p. 394–397.

[DOW 84] DOWLING W., GALLIER J., « Linear-time Algorithms for Testing the Satisfiablity of Propositional Horn Formulae », *J. Logic Programming*, vol. 3, 1984, p. 267–284, Elsevier Science Publisher.

[DUB 96] DUBOIS O., ANDRÉ P., BOUFKHAD Y., CARLIER J., « SAT versus UNSAT », *SAT Chalenge*, vol. 26, p. 415–436, AMS, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, 1996.

[EVE 76] EVEN S., ITAI A., SHAMIR A., « On the Complexity of Timetable and Multicommodity Flow Problems », *SIAM J. Comput.*, vol. 5, 1976, p. 691–703.

[FRE 78] FREUDER E., « Synthesizing Constraint Expressions », *Communications of the ACM*, vol. 21, n° 11, 1978, p. 958–966.

[FRE 82] FREUDER E., « A Sufficient Condition for Backtrack-Free Search », *Journal of the ACM*, vol. 29, n° 1, 1982, p. 24–32.

[GAL 88] GALLO G., SCUTELLÀ M., « Polynomially Solvable Satisfiability Problems », *Information Processing Letters*, vol. 29, 1988, p. 221–227, North-Holand.

[GAR 79] GAREY M., JOHNSON D., *Computers and Intractability: A Guide to the Theory of NP-Completeness*, Freeman, San Fransisco, 1979.

[GÉN 96a] GÉNISSON R., « À propos d'énumération et de polynomialité sur le problème SAT et les Problèmes de Satisfaction de Contraintes », PhD thesis, Université de Provence – Aix-Marseille I, janvier 1996.

[GÉN 96b] GÉNISSON R., JÉGOU P., « Davis and Putnam were already Checking Forward », *Proceedings of the* $13th$ *European Conference on Artificial Intelligence, ECAI'96*, Wiley, 1996, p. 180–184.

[GHA 91] GHALLAB M., ESCALADA-IMAZ E., « A linear control algorithm for a class of rule-based systems », *Journal of Logic Programming*, vol. 11, 1991, p. 117–132.

[GOE 92] GOERDT A., « A threshold for unsatifiability », HAVEL I., KOUBEK V., Eds., *Proceedings of Mathematical Foundations of Computer Science, MFCS'92*, August 1992, p. 264–272.

[HAN 88] HAN C., LEE C., « Comments on Mohr and Henderson's Path Consistency Algorithm », *Artificial Intelligence*, vol. 36, 1988, p. 125–130, Elsevier.

[HÉB 94] HÉBRARD J.-J., « A linear algorithm for renaming a set of clauses as a Horn set », *Theoretical Computer Science*, vol. 124, 1994, p. 343–350, Elsevier.

[HÉB 95] HÉBRARD J.-J., « Unique Horn renaming and Unique 2-Satisfiability », *Information Processing Letters*, vol. 54, 1995, p. 235–239, Elsevier.

[HOO 95] HOOKER J., VINAY V., « Branching Rules for Satisfiability », *Journal of Automated Reasoning*, vol. 15, 1995, p. 359–383.

[JEA 88] JEANNICOT S., OXUSOFF L., RAUZY A., « Évaluation Sémantique en Calcul Propositionnel », *Revue d'Intelligence Artificielle*, vol. 2, 1988, p. 41–60, Hermes.

[KLE 89] DE KLEER J., « A Comparison of ATMS and CSP Techniques », *Proceedings IJCAI'89*, 1989.

[KLE 99] KLEINEBUNING H., LETTMAN L., *Proposition Logic: Deduction and Algorithms*, Cambridge University Press, 1999, ISBN 0521630177.

[KNU 90] KNUTH D., « Nested Satisfiability », *Acta Informatica*, vol. 28, 1990, p. 1–6, Springer Verlag.

[LAT 88] LATOUR B., *La vie de laboratoire*, Éditions la découverte, 1988.

[LAT 89] LATOUR B., *La science en action*, Éditions la découverte, 1989.

[LEW 78] LEWIS H., « Renaming a Set of Clauses as a Horn Set », *JACM*, vol. 25, n° 1, 1978, p. 134–135.

[LI 97] LI C., ANBULAGAN, « Heuristics Based on Unit Propagation for Satisfiability Problems », *Proceedings of 15th International Joint Conference on Artificial Intelligence, IJCAI'97*, 1997, p. 366–371.

[LIN 89] LINDHORST G., SHAHROKHI F., « On renaming a set of clauses as a Horn set », *Information Processing Letters*, vol. 30, 1989, p. 289–293.

[MAC 77] MACKWORTH A., « Consistency in Networks of Relations », *Artificial Intelligence*, vol. 8, n° 1, 1977, p. 99–118.

[MAN 85] MANNILA H., MEHLORN K., « A fast algorithm for renaming a set of clauses as a Horn set », *Information Processing Letters*, vol. 21, 1985, p. 269–272.

[MEL 66] MELTZER B., « Theorem-proving for computers: some results on resolution and renaming », *Comp. Journal*, vol. 8, 1966, p. 493–495.

[MIN 88] MINOUX M., « LTUR: A Simplified Linear-Time Unit Resolution Algorithm for Horn Formulae and its Computer Implementation », *Information Processing Letters*, vol. 29, 1988, p. 1–12, North Holland.

[MOH 86] MOHR R., HENDERSON T., « Arc and Path Consistency Revisited », *Artificial Intelligence*, vol. 38, 1986, p. 225–233.

[MON 85] MONIEN B., SPECKENMEYER E., « Solving Satisfiability in Less than 2^n Steps », *Discrete Applied Math.*, vol. 10, 1985, p. 287–295.

[PET 91] PETRESCHI R., SIMEONE B., « Experimental Comparison on 2-Satisfiability Algorithms », *RAIRO Recherche Opérationelle*, vol. 25, 1991, p. 241–264.

[PRE 92] PRESS W., TEUKOLSKY S., VETTERLING W., FALNNERY B., Eds., *Numerical Recipes in C: the Art of Scientific Computing*, Cambridge University Press, 1988-1992, ISBN 0-521-43108-5.

[RAU 95] RAUZY A., « Polynomial restrictions of SAT: What can be done with an efficient implementation of the Davis and Putnam's procedure », MONTANARI U., ROSSI F., Eds., *Proceedings of the International Conference on Principle of Constraint Programming, CP'95*, vol. 976 de *LNCS*, Springer Verlag, 1995, p. 515–532.

[RAU 99] RAUZY A., SAIS L., BRISOUX L., « Calcul propositionnel : Vers une extension du formalisme », *Actes de Journées Nationales sur la Résolution Pratique de Problèmes NP-Complets, JNPC'99*, 1999.

[SAB 94] SABIN D., FREUDER E., « Contradicting conventional wisdom in constraint satisfaction », BORNING A., Ed., *Proceedings of the Second Workshop on Principles and Practice of Constraint Programming PPCP'94*, Seattle WA, May 1994.

[VAL 79] VALIANT L., « The complexity of enumeration and reliability problems », *SIAM Journal of Computing*, vol. 8, 1979, p. 410-421.

[VIE 90] VIENNOT X., « Tree Everywhere », *Proceedings 15th CAAP*, LNCS 431, Springer Verlag, 1990, p. 18–41.