Satisfiabilité minimale et applications *

Chu-Min Li [1,2], Zhu Zhu[2], Felip Manyà[3], Laurent Simon[4]

[1]Huazhong Univ. of Science and Technology, Wuhan, China — chu-min.li@u-picardie.fr
[2]Univ. de Picardie Jules Verne Amiens, France — zhu.zhu@u-picardie.fr
[3]IIIA-CSIC Bellaterra, Spain — felip@iiia.csic.es
[4]Univ. Orsay Paris Sud 11 Orsay, France — simon@lri.fr

Abstract

Nous proposons, dans cet article, de placer le problème de satisfiabilité minimale (MinSAT) sur le devant de la scène. Pour cela, un algorithme efficace de type *branch-and-bound* pour résoudre le problème MinSAT partiel avec pondération est introduit, permettant du même coup une évaluation empirique de cet algorithme pour les cas Min-3SAT aléatoires, MaxClique et pour les problèmes d'enchères combinatoires. Les techniques employées pour résoudre MinSAT se démarquent de celles utilisées pour le problème de satisfiabilité maximale (MaxSAT), qui a lui été très étudié. Nos résultats démontrent de manière empirique que, notamment sur les problèmes d'enchères combinatoires, une réduction du problème initial à MinSAT peut être plus intéressante qu'une réduction à MaxSAT, ou même que l'utilisation de méthodes dédiées. Nous étendons par ailleurs ce travail en montrant une corrélation entre le nombre minimal et le nombre maximal de clauses satisfaites dans une instance SAT aléatoire donnée.

1 Introduction

La résolution pratique de problèmes NP-Complets, via une réduction au problème SAT de la logique propositionnelle, est une technique puissante qui a permis, et permet encore, de résoudre des problèmes académiques ainsi que des problèmes d'importance industrielle. Plus récemment, les succès obtenus autour de SAT ont permis d'envisager des formalismes liés, comme MaxSAT, MaxSAT partiel ou encore MaxSAT partiel pondéré [Li and Manyà 2009]. Les résultats obtenus, plus que prometteurs, montrent combien ils deviennent des approches génériques crédibles pour résoudre, en pratique, des problèmes d'optimisations, à l'instar de SAT et des problèmes de décision.

Dans cet article, nous proposons de prendre une nouvelle direction. Plutôt que de se focaliser sur le formalisme MaxSAT, qui tente de maximiser le coût des clauses satisfaites, nous allons nous concentrer sur le problème opposé, c'est à dire tenter de minimiser le coût des clauses satisfaites. De manière plus précise, nous nous focalisons sur le problème MinSAT partiel pondéré, dans lequel les instances sont représentées par des ensembles de clauses, chaque clause étant déclarée dure (*hard*) ou molle (*soft*), et ayant éventuellement un poids particulier. Résoudre le problème MinSAT partiel pondéré revient à trouver une assignation des variables qui satisfasse toutes les clauses dures, et qui minimise la somme des poids des clauses molles. Ce problème de minimisation est intéressant pour, notamment, deux raisons : (1) Certains problèmes admettent un encodage plus naturel en les considérant comme des problèmes de minimisation plutôt que des problèmes de maximisation, (2) Chercher à minimiser va nous permettre d'introduire une nouvelle technique, relativement puissante, de bornage supérieur, qui ne peut être appliquée aux solveurs MaxSAT basés sur les techniques branch-and-bound. Alors que les deux problèmes MaxSat et MinSat sont tous deux des extensions naturelles de SAT, les techniques que l'on peut mettre en œuvre pour les résoudre sont bien différentes en pratique.

Plus concrètement, nous proposons dans la suite de cet article un algorithme de type branch-and-bound pour le problème MinSAT pondéré partiel, intégrant de nouvelles techniques de bornage. Cet algorithme, appelé $MinSatz$ est également testé grâce à une importante étude empirique. Celle-ci prend en compte les problèmes aléatoires Min-3SAT, MaxClique ainsi que les problèmes d'enchères combinatoires. Les résultats obtenus démontrent combien une réduction à MinSAT peut être plus efficace qu'une réduction à MaxSAT sur ces problèmes, y compris en com-

*Ce travail est soutenu par le projet français ANR UNLOC (ANR-08-BLAN-0289-03), par les projets espagnols Consolider-Ingenio10 CSD2007-0022, TIN2010-20967-C04-01, SGU P. N. de Movilidad de Recursos Humanos, ainsi que par les projets GenCat 2009-SGR-1434, et par le National Natural Science Foundation of China (NSFC) grant No. 61070235.

parant nos résultats avec d'autres méthodes dédiées du domaine. L'étude du problème MinSAT nous permet aussi de publier ici une corrélation intrigante entre la valeur MinSAT et la valeur MaxSAT sur les problèmes 3SAT.

À notre connaissance, cet article introduit le premier solveur exact spécifique pour les instances MinSAT partielles pondérées (et les variantes MinSAT, MinSAT pondérées et MinSAT partielles). C'est aussi la première fois que le formalisme MinSAT est utilisé, avec succès, pour résoudre des problèmes concrets d'optimisation. Ce travail a été précédé de, notamment, [Li *et al.* 2010b], dans lequel les auteurs proposaient différents moyens d'encoder le problème MinSAT (non pondéré) en un problème MaxSAT. Cette réduction ne permettait pas une généralisation aux instances MinSAT partielles pondérées qui nous intéressent ici. Par ailleurs, les expérimentations étaient limitées aux problèmes aléatoires. Notons aussi que, même s'il n'avait jusqu'à présent été étudié que de manière parcellaire (par rapport au problème MaxSAT), le problème MinSAT (non pondéré) a tout de même été étudié dans quelques travaux (voir par exemple [Avidor and Zwick 2005, Kohli *et al.* 1994, Marathe and Ravi 1996] et les références incluses).

La suite de cet article est structuré de la manière suivante : la section 2 contient les définitions de base en graphe et autour du problème MinSAT. La section 3 décrit le solveur MinSAT que nous proposons, accompagné d'une description de ses techniques de bornage. La section 4 présente les problèmes et les solveurs utilisés dans l'évaluation empirique, ainsi qu'une discussion des résultats obtenus. Bien entendu, la section 5 conclut l'article en proposant notamment quelques directions de recherche possibles.

2 Préliminaires et notations

Un littéral est une variable propositionnelle ou sa négation. Une clause est une disjonction de littéraux. Une clause pondérée est une paire (c, w), où c est une clause et w, son poids, est un entier naturel ou infini. Une clause est dure si son poids est l'infini ; sinon est appelée molle. Une instance de MinSAT partiel pondéré (idem pour MaxSAT) est un multi-ensemble de clauses pondérées $\phi = \{(h_1, \infty), \ldots, (h_k, \infty), (c_1, w_1), \ldots, (c_m,\ w_m)\}$, où les premières k clauses sont dures et les dernières m clauses sont molles. Par soucis de simplicité, dans la suite, nous omettrons de spécifier les poids infinis, et écrirons plutôt $\phi = \{h_1, \ldots, h_k, (c_1, w_1), \ldots, (c_m, w_m)\}$.

Une interprétation est une assignation de chaque variable à 0 ou 1. Le coût d'une interprétation l de ϕ est la somme des coûts des clauses satisfaites par l. Le problème MinSAT partiel pondéré, pour une instance ϕ, revient à trouver une interprétation des variables de coût minimum qui satisfasse toutes les clauses dures (i.e. une assignation optimale), tandis que le problème MaxSAT partiel pondéré revient quant à lui à trouver l'assignation de coût maximum satisfaisant toutes les clauses dures. De manière exhaustive nous pouvons définir tous les sous-problèmes : le problème MinSAT (MaxSAT) partiel est le problème MinSAT (MaxSAT) partiel pondéré dans lequel toutes les clauses molles sont de poids 1. Le problème MinSAT (MaxSAT) est le problème MinSAT (MaxSAT) partiel dans lequel il n'y a pas de clauses dures. Le problème SAT est le problème MaxSAT partiel dans lequel il n'y a pas de clauses molles.

Une *clique* d'un graphe non dirigé $G = (V, E)$, où V est l'ensemble des noeuds et E l'ensemble des arcs, est un sous-ensemble C de V tel que, pour chaque paire de noeuds de C, il existe un arc qui les connecte. Cela revient à dire que le sous-graphe induit par C est complet. Une clique maximale est une clique de la plus grande taille possible. Le *problème de clique maximale (MaxClique)* pour un graphe non dirigé G revient à trouver une clique maximale dans G. Une *partition en cliques* d'un graphe non dirigé $G = (V, E)$ est une partition de V en ensembles disjoints $V_1, \ldots, V_k$ telle que, pour $1 \leq i \leq k$, le sous-graphe induit par V_i est un graphe complet. Soit $\chi(G)$ le nombre minimum de couleurs nécessaires pour colorier les noeuds de G de façon à ce que deux noeuds adjacents soient de couleurs différentes, et soit $\omega(G)$ la taille de la clique maximale de G, on dit que G est parfait si $\chi(G')$=$\omega(G')$ pour tout sous-graphe induit G' de G.

3 Un solveur MinSAT exact

Nous introduisons dans cette section le premier solveur spécifique pour les instances MinSAT partielles pondérées (à notre connaissance), appelé MinSatz, basé sur Satz [Li and Anbulagan 1997], qui a d'ailleurs aussi servi de base à MaxSatz [Li *et al.* 2007]. Pour des raisons de simplicité nous commencerons par présenter le cas MinSAT partiel (non pondéré). MinSatz se base sur le schéma classique branch-and-bound, dont l'espace de recherche est formé par un arbre représentant toutes les interprétations possibles. Contrairement à un solveur MaxSAT de type branch-and-bound, comme MaxSatz, qui résout les problèmes MaxSAT en minimisant le nombre de clauses non satisfaites, MinSatz doit quant à lui maximiser le nombre de clauses falsifiées.

A chaque nœud de l'arbre de recherche, MinSatz commence par appliquer la propagation unitaire sur les seules clauses dures (i.e. étant donnée une clause dure l, initiale ou nouvellement dérivée, MinSatz satisfait et retire toutes les clauses contenant le littéral l, et efface toutes les occurrences de $\neg l$; les clauses molles ne sont pas propagées dans la mesure où elles ne sont pas forcément satisfaites). Si l'une des clauses dures devient vide, MinSatz revient sur ses choix précédents. Sinon, il calcule une borne supérieur du nombre maximal des clauses molles falsifiées (UB) si l'assignation partielle courante est étendue jusqu'à devenir

complète. Ce nombre, UB, est comparé avec le nombre de clauses falsifiées, associé à la meilleur assignation découverte jusque là (LB). Si $UB \leq LB$, une meilleure solution ne peut pas être trouvée à partir du nœud courant, et MinSatz doit aussi revenir en arrière. Sinon, une variable est sélectionnée et instanciée. Tant que l'espace de recherche n'a pas été totalement exploré, MinSatz répète ce procédé. Lorsque la recherche est finie, MinSatz retourne la meilleur solution trouvée. L'algorithme 1 détaille le pseudo-code de MinSatz.

Algorithm 1: MinSatz(ϕ, LB)

$\phi \leftarrow$ hardUnitPropagation(ϕ) ;
Si ϕ contient une clause dure vide **Alors Retourner** -1 ;
Si ϕ={} **Ou** ϕ ne contient que des clauses molles vides
Alors Retourner #empty(ϕ) ;
UB $\leftarrow$ #empty(ϕ)+overestimation(ϕ) ;
Si (UB $\leq$ LB) **Alors Retourner** LB ;
x $\leftarrow$ select(ϕ) ;
LB $\leftarrow$ MinSatz(ϕ_x, LB) ;
LB $\leftarrow$ MinSatz($\phi_{\neg x}$, LB) ;
Retourner LB.

Pour résoudre une instance ϕ, il faut appeler MinSatz(ϕ, 0). Si MinSatz retourne -1, cela signifie que la partie dure des clauses de ϕ est insatisfiable, et il n'y a pas de solution possible. La fonction #empty(ϕ) retourne le nombre de clauses molles vides dans ϕ ; overestimation(ϕ) retourne une surestimation du nombre de clauses molles qui seraient insatisfiables si l'assignation partielle courante était étendue jusqu'à être complète. select(ϕ) retourne la variable la plus fréquente dans ϕ. L'instance ϕ_x (resp. $\phi_{\neg x}$) est ϕ dans laquelle toutes les clauses contenant x (resp. $\neg x$) sont satisfaites et retirées, et dans laquelle le littéral $\neg x$ (resp. x) est retiré des clauses restantes. La valeur MinSAT de l'instance initiale ϕ (i.e. le nombre minimal de clauses satisfaites de ϕ) est #soft(ϕ)-MinSatz(ϕ, 0), où #soft(ϕ) est le nombre de clauses molles de ϕ.

Prenons le temps maintenant de décrire comment est calculé UB. Supposons que l'on se trouve sur un nœud de l'espace de recherche et que, après avoir appliqué la propagation unitaire sur les clauses dures uniquement, on se trouve devant une instance formée des clauses dures $\{h_1, \ldots, h_k\}$, et des clauses molles $\{c_1, \ldots, c_m\}$ (non encore décidées). On construit alors un graphe non dirigé $G = (V, E)$ où V contient un nœud pour chaque clause dans $\{c_1, \ldots, c_m\}$ (que l'on notera $V = \{v_1, \ldots, v_m\}$). Un arc est ajouté entre v_i et v_j si la clause c_i contient un littéral l, et si la clause c_j contient $\neg l$.

De plus, un arc entre v_i et v_j est ajouté dès que l'on peut prouver, par propagation unitaire, la contradiction sur l'ensemble des littéraux $\{\neg l^i_1, \ldots, \neg l^i_p, \neg l^j_1, \ldots, \neg l^j_q, h_1, \ldots, h_k\}$, avec $c_i = \{l^i_1, \ldots, l^i_p\}$ (correspondant au nœud v_i) et $c_j = \{l^j_1, \ldots, l^j_q\}$ (associé au nœud v_j). Intuitivement, la représentation à l'aide du graphe G permet d'exprimer le fait que les clauses molles associées à deux arcs d'un même nœud ne peuvent pas être simultanément falsifiées. Dans le premier cas, il existe en effet au moins une clause molle satisfaite dans la mesure ou soit l, soit $\neg l$ est satisfait par toute assignation. Dans le second cas, si les clauses molles associées à deux arcs d'un nœud sont falsifiées, alors, par construction, cela impliquerait la falsification d'une clause dure.

Une fois que le graphe G est construit, une partition en cliques est construite à l'aide de l'algorithme décrit dans [Li and Quan 2010b, Li and Quan 2010a]. Par construction de G, il existe au plus une clause falsifiée dans chaque clique, et au moins toutes les clauses de chaque clique, sauf une, sont satisfaites par toute assignation complète. Le nombre de cliques d'une partition, noté s, est donc une surestimation du nombre de clauses molles qui peuvent être falsifiées par l'assignation partielle courante, par rapport à toute extension complète de celle-ci. En se basant sur ce point, on peut définir $UB = e + s$, où e est le nombre de clauses molles vides dans l'instance MinSAT courante.

On peut noter que le graphe aurait aussi pu être défini de telle manière qu'un arc entre v_i et v_j aurait été ajouté dès que l'ensemble de clauses $\{\neg l^i_1, \ldots, \neg l^i_p, \neg l^j_1, \ldots, \neg l^j_q, h_1, \ldots, h_k\}$ aurait pu être prouvé insatisfiable en utilisant un SAT solveur complet comme une boîte noire, plutôt qu'en se basant sur la propagation unitaire. Cela aurait certainement conduit à de meilleurs calculs de bornes mais aurait aussi, en contrepartie, eu un impact sur la rapidité du solveur. Nous nous sommes donc restreint dans ce travail aux contradictions détectables par propagation unitaire.

Exemple 1 *Supposons que, lors de la recherche, l'on soit sur un nœud contenant les clauses dures* $\neg x_1 \vee \neg x_2$, $\neg x_2 \vee \neg x_3$, $\neg x_3 \vee \neg x_4$, $\neg x_4 \vee \neg x_5$, $\neg x_1 \vee \neg x_5$, *et les clauses molles* $\neg x_1, \neg x_2, \neg x_3, \neg x_4, \neg x_5$. *Supposons, de plus, qu'aucune clause n'a encore été réduite à vide. On construit alors le graphe* G *ayant les nœuds* v_1, v_2, v_3, v_4, v_5, *chaque nœud* v_i *étant associé aux clauses molles* $\neg x_i$ *(pour* $1 \leq i \leq 5$*). Les arcs de* G *sont* $\{(v_1, v_2), (v_2, v_3), (v_3, v_4), (v_4, v_5), (v_1, v_5)\}$, *ce qui correspond au graphe représenté figure 1. Supposons maintenant que notre algorithme trouve la partition en cliques de* G *suivante :* $\{\{v_1, v_2\}, \{v_3, v_4\}, \{v_5)\}\}$. *Alors, au plus 3 clauses molles sur les 5 clauses initiales peuvent être réduites à vide, ce qui nous donne* $UB = 3$.

Cependant, il faut bien noter que la partition ne peut donner une borne supérieure optimale si G n'est pas parfait, comme l'exemple suivant, adapté de celui donné dans [Li and Quan 2010b], le montre bien.

En effet, le graphe figure 1 peut être partitionné en trois cliques $\{(v_1, v_2), (v_3, v_4), (v_5)\}$, donnant dès

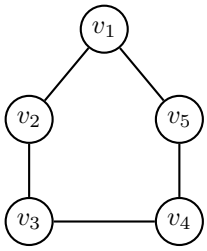

FIG. 1 – Un graphe imparfait simple ($\chi(G)$=3 et $\omega(G)$=2)

lors une borne supérieure de 3 pour le nombre de clauses falsifiées. Néanmoins, une analyse plus poussée montre que seulement deux clauses peuvent être falsifiées (au lieu de 3). Cette analyse peut par exemple se baser sur une adaptation de l'approche proposée dans [Li and Quan 2010b, Li and Quan 2010a] pour transformer la partition en cliques de G en une instance MaxSAT partielle. Dès lors, il devient possible d'utiliser un solveur MaxSAT pour améliorer la borne supérieure. Le raisonnement, en logique propositionnelle, de ce solveur serait équivalent au suivant : Supposons que toute clique contienne une clause falsifiée pour toute assignation complète. Alors, v_5 devrait être falsifié, mais v_1 et v_4 ne le peuvent, puisque v_1 et v_4 sont connectés à v_5. Donc, la seule possibilité pour la première et la seconde clique de contenir une clause falsifiée serait que v_2 et v_3 soient tous deux falsifiés, ce qui est impossible dans la mesure où v_2 et v_3 sont connectés. Donc, $\{(v_1, v_2), (v_3, v_4), (v_5)\}$ est un sous-ensemble de cliques qui ne permet pas à toutes ses cliques d'avoir une clause falsifiée en même temps.

Dans le cas général, on utilise les techniques liées à MaxSAT pour détecter des sous-ensembles de cliques qui ne peuvent avoir en même temps des clauses falsifiées, comme dans [Li and Quan 2010b, Li and Quan 2010a]. Chaque sous-ensemble permet d'améliorer la borne supérieure de un. Le lecteur est invité à se référer aux deux articles précédemment cités pour de plus amples détails sur ces techniques. La détection demande un temps en $O(m^2)$ pour la partition de graphe (pour m clauses molles), ce qui induit une complexité en temps de la fonction overestimation(ϕ) en $O(m^2)$ également. La représentation du graphe demande par ailleurs un espace en $O(m^2)$ et les cliques obtenues sont stockées dans un espace en $O(m)$.

Dans le cas pondéré, lorsque tous les poids sont des entiers positifs, on définit le poids d'un nœud v de G comme étant le poids de la clause molle associée c. Soit $\mathcal{P}$ l'ensemble des cliques de G, tel que deux cliques ont la possibilité de partager un nœud, et tel que chaque clique clq_i est associée à un poids w_{clq_i}. On appelle $\mathcal{P}$ la *Partition pondérée en cliques* de G si pour tout nœud v de G, son poids est égal à $\sum_{v \in clq_i} w_{clq_i}$. Par définition, chaque nœud de G appartient à au moins une clique de $\mathcal{P}$.

Exemple 2 *Dans la figure 1, supposons que les cinq clauses molles pondérées correspondant aux cinq noeuds soient {(c_1, 2), (c_2, 3), (c_3, 4), (c_4, 5), (c_5, 6)}. Alors $\mathcal{P}_1$={{(v_1, v_2), 2}, {(v_3, v_4), 4}, {(v_5), 6}, {(v_2), 1}, {(v_4), 1}} et $\mathcal{P}_2$={{(v_1, v_2), 2}, {(v_2, v_3), 1}, {(v_3, v_4), 3}, {(v_4, v_5), 2}, {(v_5), 4}} sont des partitions pondérées en cliques du graphe.*

Soit une instance MinSAT ϕ, une borne supérieure triviale pour la somme des poids des clauses molles falsifiées est $UB = \sum_{c_i \in \phi} w_i$, où w_i est le poids de la clause c_i. Chaque clique $\{(v_{i_1}, \ldots, v_{i_k}), w\}$ de poids w permet d'améliorer cette borne de (k-1)w, dans la mesure où cette clique implique au moins k-1 clauses satisfaites. En d'autres termes, chaque clique permet de ne pas compter, dans la borne supérieure, le poids w des k-1 clauses molles. Dans l'exemple 2, la partition pondérée en cliques $\mathcal{P}_1$ permet d'obtenir une borne supérieure de 14, alors que $\mathcal{P}_2$ donne une borne supérieure de 12.

La génération d'une partition pondérée en cliques d'un graphe pondéré G revient à partitionner les poids de tous les nœuds en un ensemble de cliques de G. C'est ce que permet de faire l'algorithme 2, utilisé dans MinSatz. Tout d'abord, cet algorithme partitionne G sans prendre en compte les poids associés. Ensuite, il cherche à associer à chaque clique $\{v_{i_1}, \ldots, v_{i_k}\}$ le poids $w = \min(w_{i_1}, \ldots, w_{i_k})$, où w_{i_j} est le poids de v_{i_j} dans G pour $1 \leq j \leq k$. Il modifie ensuite le poids de v_{i_j} dans G pour qu'il soit $w_{i_j} - w$. Une fois que toutes les cliques ont été associées à un poids, un nouveau graphe G' est construit d'après G, en ôtant tous les nœuds de poids nul. Ce mécanisme se poursuit ensuite pour G'. On peut noter que s'il y a t cliques dans la partition de G, G' a au moins t nœuds de moins que G.

Algorithm 2: partition(ϕ)

```
Construit un graphe pondéré G associé à φ ;
P ← { } ;
repeat
    Trouve une partition en cliques de G, et ajoute les
    cliques à P ;
    Construit G' d'après les G et les cliques obtenues ;
    G ← G' ;
until G devienne vide
Retourner P.
```

Dans l'exemple 2, $\mathcal{P}_1$ est obtenu en utilisant l'algorithme 2. On notera que $\mathcal{P}_2$ est de meilleur qualité que $\mathcal{P}_1$. Bien entendu, trouver de manière efficace une partition pondérée en cliques de meilleur qualité va permettre d'améliorer de manière significative la recherche restant à faire.

L'utilisation des technologies MaxSAT va aussi permettre d'améliorer encore la borne supérieure donnée par

la partition ci-dessus, en suivant la même idée que pour le cas non pondéré. Chaque fois que l'on détecte un sous-ensemble de cliques dans lequel toutes les cliques ne peuvent avoir une clause falsifiée en même temps, la borne supérieure peut être améliorée de w, où w est le poids minimum sur toutes les cliques du sous-ensemble.

4 Évaluation expérimentale

L'évaluation expérimentale tient une part importante dans notre travail. Nous avons ainsi conduit cinq expérimentations pour nous permettre d'évaluer les performances et l'utilité pratique de MinSatz. Les expérimentations, sauf mention contraire, ont été réalisées sur un Macpro 2.8GHz Intel Xeon ayant 4Go de mémoire sous Mac OS X 10.5.

4.1 Benchmarks

MaxClique. L'encodage en problème MaxSAT partiel du problème MaxClique d'un graphe $G = (V, E)$, tel qu'utilisé dans les évaluations MaxSAT est le suivant [Heras *et al.* 2008] : Une variable propositionnelle est associée à chaque nœud de V. La variable x_i est vraie ssi le nœud v_i appartient à la clique. Pour chaque paire v_i, v_j de nœuds non adjacents, correspond une clause dure $\neg x_i \vee \neg x_j$. Pour chaque nœud v_i, il y a une clause molle unitaire $(x_i, 1)$. Résoudre l'instance correspondante revient à chercher à maximiser le nombre de nœuds appartenant à la clique.

On peut obtenir très simplement un encodage MinSAT partiel de ce même problème MaxClique en utilisant les mêmes clauses dures et, pour chaque nœud v_i, en utilisant plutôt une clause molle unitaire $(\neg x_i, 1)$. Dès lors, résoudre l'instance obtenue revient à minimiser le nombre de nœuds n'appartenant pas à la clique.

Enchères combinatoires. Pour rappel, le problème des enchères combinatoires se définit comme suit. On a un ensemble de biens G, et un ensemble d'acheteurs qui peuvent enchérir sur des ensembles indivisibles de biens. Chaque enchère b_i est définie comme le sous-ensemble des biens voulus $G_i \subseteq G$ et le montant proposé pour l'achat. Le vendeur, qui veut bien entendu maximiser ses revenus, doit décider quelles enchères il désire satisfaire, dans la mesure où deux enchères ayant un bien commun ne peuvent être simultanément acceptées. L'encodage en problème MaxSAT partiel pondéré de ce problème, tel que proposé dans l'évaluation MaxSAT est le suivant [Heras *et al.* 2008] : Une variable propositionnelle est associée à chaque enchère. x_i est vraie ssi l'enchère b_i est acceptée. Pour chaque paire d'enchères (b_i, b_j) contenant au moins un bien commun, une clause dure $\neg x_i \vee \neg x_j$ est ajoutée (encodant ainsi le fait qu'on ne puisse pas accepter ces deux enchères simultanément). Pour chaque enchère b_i, est ajouté une clause unitaire molle (x_i, w_i) représentant le profit w_i potentiel associé à l'acceptation de l'enchère b_i. Ainsi, résoudre l'instance revient à maximiser les profits obtenus.

L'encodage sous forme de problème MinSAT partiel pondéré s'obtient en utilisant les mêmes clauses dures que précédemment, mais en considérant plutôt les pertes de profits associées au refus de chaque enchère. Ainsi, pour chaque enchère b_i est ajouté une clause unitaire molle $(\neg x_i, w_i)$ indiquant une perte de profit de w_i si b_i n'est pas acceptée. Dans ce cas, résoudre l'instance revient naturellement à minimiser les pertes.

Min-3SAT. Rappelons que le problème Min-3SAT (non pondéré) consiste à résoudre une instance MinSAT dont chaque clause contient exactement 3 littéraux. MinSAT peut aussi se réduire à MaxSAT partiel. Trois encodages (E1, E2, et E3) ont été présentés dans [Li *et al.* 2010b].

4.2 Solveurs

Nous avons comparé MinSatz aux solveurs suivants :

- akmaxsat, akmaxsat_ls [Kuegel 2010], Inc-MaxSatz [Lin *et al.* 2008], MaxSatz, Wmax-satz+ [Li *et al.* 2007, Li *et al.* 2010a] : solveurs MaxSAT partiel pondéré, basés sur des techniques branch-and-bound, et soumis à l'évaluation MaxSAT 2010.
- MaxCLQ [Li and Quan 2010a] et MaxCliqueDyn (Noté MCQDyn) [Konc and Janezic 2007] : les deux meilleurs solveurs spécifiques au problème Max-Clique, à notre connaissance.
- CASS [Fujishima *et al.* 1999] : solveur spécifique (état de l'art) pour le problème des enchères combinatoires.
- CPLEX : solveur commercial (version 8.0) utilisant des techniques de programmation linéaire.

4.3 Analyse des résultats expérimentaux

Toutes les tables reportent des temps d'exécutions moyens en secondes, calculés sur celles ayant terminé avant la limite de temps donnée. Ces temps sont donc suivis du nombre d'instances effectivement résolues.

Dans la première expérimentation, nous avons cherché à résoudre les problèmes Min-3SAT aléatoires de [Li *et al.* 2010b], en utilisant une limite de temps de 3 heures (identique à [Li *et al.* 2010b]). Nous comparons ici les performances de MinSatz avec l'encodage en MaxSat partiel proposé dans l'article cité. Les résultats sont résumés table 1. Notons que ces instances ont été générées uniformément, de 40 à 100 variables, et avec un nombre de clauses correspondant à des ratios clauses-sur-variables (C/V) prédéterminés (4, 4.25 et 5).

La table 1 montre clairement combien un solveur MinSAT dédié est plus performant que le meilleur solveur MaxSAT. On notera que MinSatz a résolu toutes les ins-

$instance$		$MinSatz$	$MaxSatz$		
#var	C/V		$E1$	$E2$	$E3$
40	4.00	0.01*(50)*	3.28*(50)*	507.8*(50)*	0.19*(50)*
50	4.00	0.03*(50)*	49.30*(50)*	- *(0)*	0.92*(50)*
60	4.00	0.12*(50)*	742.4*(50)*	- *(0)*	4.96*(50)*
70	4.00	0.41*(50)*	5735*(34)*	- *(0)*	23.21*(50)*
80	4.00	1.97*(50)*	- *(0)*	- *(0)*	100.7*(50)*
90	4.00	8.87*(50)*	- *(0)*	- *(0)*	381.5*(50)*
100	4.00	30.59*(50)*	- *(0)*	- *(0)*	1658*(33)*
40	4.25	0.01*(50)*	4.67*(50)*	992.5*(50)*	0.28*(50)*
50	4.25	0.04*(50)*	75.6*(50)*	- *(0)*	1.57*(50)*
60	4.25	0.14*(50)*	1153*(50)*	- *(0)*	8.31 *(50)*
70	4.25	0.69*(50)*	5989*(5)*	- *(0)*	42.77*(50)*
80	4.25	3.02*(50)*	- *(0)*	- *(0)*	186.3*(50)*
90	4.25	10.52*(50)*	- *(0)*	- *(0)*	760.4*(50)*
100	4.25	61.85*(50)*	- *(0)*	- *(0)*	2819*(26)*
40	5.00	0.02*(50)*	10.87*(50)*	5693*(48)*	0.80*(50)*
50	5.00	0.07*(50)*	226.8*(50)*	- *(0)*	5.24*(50)*
60	5.00	0.47*(50)*	3803*(48)*	- *(0)*	39.3*(50)*
70	5.00	1.59*(50)*	- *(0)*	- *(0)*	243.4*(50)*
80	5.00	14.68*(50)*	- *(0)*	- *(0)*	1512*(50)*
90	5.00	75.57*(50)*	- *(0)*	- *(0)*	5167*(39)*
100	5.00	380.72*(50)*	- *(0)*	- *(0)*	- *(0)*

TAB. 1 – Temps moyens (en secondes) de résolution pour les instances Min-3SAT résolues en 3 heures, suivi par le nombre d'instances effectivement résolues (sur 50)

tances dans le temps imparti, ce qui n'est pas le cas de MaxSatz.

Dans l'expérimentation suivante, nous avons cherché à résoudre les 96 problèmes aléatoires de MaxClique utilisés dans l'évaluation MaxSAT 2010[1], ainsi que les 66 [2] instances de DIMACS (dont 62 sont aussi dans l'évaluation sous le nom *structured*). Le temps maximal est aligné sur celui de l'évaluation (1800s). Nous comparons MinSatz avec les deux meilleurs solveurs MaxSAT sur ces instances : IncMaxSatz et akmaxsat, ainsi qu'avec les deux meilleurs solveurs spécifiques pour MaxClique (MaxCLQ et MCQDyn). Cette expérimentation a été conduite sur une machine Intel core 2 duo à 3.33 GHz de 4Go de mémoire sous Linux, dans la mesure où les solveurs MaxSAT étaient disponibles en binaires pour ce système seulement. Les résultats sont résumés table 2. Clairement, MinSatz obtient des performances bien supérieures aux deux solveurs MaxSAT et arrive à égaliser les performances des solveurs spécifiques. Dans la mesure où le problème MaxClique, en soit, est étudié depuis longtemps, nous pensons que cela montre qu'une réduction à MinSAT est une solution alternative pour toute une série de problèmes d'optimisations.

Dans la troisième expérimentation, nous avons chercher à résoudre les problèmes d'enchères combinatoires utilisées dans l'évaluation MaxSAT 2010. Ils ont été générés à partir de la suite de problèmes *Combinatorial Auction*

[1] http ://www.maxsat.udl.cat/10/introduction/index.html
[2] available at http ://cs.hbg.psu.edu/txn131/clique.html

instances	MinSatz	IncMaxsatz	akmaxsat	MaxCLQ	MCQDyn
random(96)	0.01*(96)*	1.6*(96)*	3*(96)*	0.01*(96)*	0.01*(96)*
DIMACS(66)	115*(53)*	88*(39)*	84*(40)*	69*(54)*	57*(52)*

TAB. 2 – Temps moyens (en secondes) pour MaxClique, dans un temps maximal de 1800s, suivi par le nombre d'instances résolues.

Test Suite (CATS) [Leyton-Brown *et al.* 2000], basée sur un générateur aléatoire s'inspirant d'exemples réels. Nous avons utilisés les ensembles *Paths* (88 instances) et *Scheduling* (84 instances). Encore une fois, nous comparons MinSatz avec les résultats obtenus lors de l'évaluation MaxSAT 2010, mais avec les solveurs akmaxsat, akmaxsat_ls, et Wmaxsatz+, qui sont les meilleurs sur ces instances. Le temps maximal est de 1800s. Les résultats en temps de ces solveurs sont ceux de l'évaluation, obtenus sur une machine AMD Opteron de 2GHz, ayant 512Mo de mémoire, sous Linux. Les résultats sont donnés table 3. Même en prenant en compte la différence de puissance entre les machines utilisées pour les tests, il demeure clair que MinSatz est de plusieurs ordres de grandeur plus performants que ces solveurs MaxSAT. Dans la mesure où ces instances étaient finalement facile pour MinSatz, nous avons décidé de comparer notre approche avec les meilleurs solveurs dédiés du domaine.

instances	MinSatz	akmaxsat_ls	akmaxsat	Wmaxsatz+
path (88)	0.01*(88)*	22*(88)*	23*(88)*	192*(71)*
scheduling (84)	0.01*(84)*	267*(75)*	230*(73)*	63*(83)*

TAB. 3 – Problèmes d'enchères combinatoires (temps en secondes)

Ceci nous mène donc à notre quatrième expérimentation, où nous comparons les performances de MinSatz avec CASS et CPLEX sur les instances d'enchères combinatoires disponibles à http ://people.cs.ubc.ca/~ kevinlb/downloads.html. Ces instances sont générées par CATS 2.0 pour 9 distributions. Nous avons utilisé la suite "variable problem size" ayant de 40 à 400 biens et de 50 à 2000 enchères. Chaque distribution contient 800 instances, mais, par soucis d'économie, nous avons arbitrairement restreint notre étude aux 100 premières instances (numérotées 000001 à 000100). Les résultats sont reproduits table 4, en utilisant cette fois un temps limite de 3600s. Les temps de calcul de CASS et de CPLEX pour ces instances, obtenus sur une machine Xeon 2.4GHz, sont reproduits depuis cette même page internet. Pour tenter de comparer au mieux les différentes approches, nous avons divisé les temps de CASS et CPLEX pour prendre en compte la différence d'architecture machine. MinSatz est plus rapide que CASS sur ces instances, mais plus lent que CPLEX. On notera cependant que, sur la distribution L7, MinSatz résout un

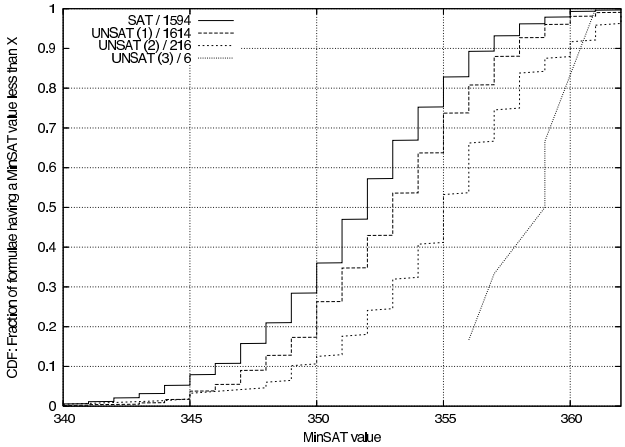

FIG. 2 – Courbes CDF pour 511 clauses et 120 variables (3430 essais). Les courbes sont respectivement basées sur 1594 instances (MaxSAT=511, i.e. toutes les 511 clauses de chaque instance peuvent être satisfaites), 1614 instances (MaxSAT=510, i.e. 510 seulement des 511 clauses peuvent être satisfaites), 216 instances (MaxSAT=509) et seulement 6 instances pour MaxSAT=508.

certain nombre d'instances que CPLEX n'arrive pas à résoudre dans le temps imparti.

Instances	MinSatz	CASS	CPLEX
Arbitrary(100)	108*(52)*	133*(39)*	34*(68)*
L2(100)	0.01*(100)*	0.01*(100)*	1*(100)*
L3(100)	164*(43)*	165*(32)*	161*(50)*
L4(100)	131*(52)*	84 *(46)*	32*(97)*
L6(100)	170*(44)*	90*(38)*	0.01*(100)*
L7(100)	56*(93)*	6*(88)*	154*(89)*
Matching(100)	98*(79)*	148*(31)*	0.01*(100)*
Region(100)	186*(50)*	37*(35)*	17*(96)*
Scheduling(100)	112*(82)*	28*(71)*	2*(100)*
Total(900)	98*(595)*	44*(480)*	49*(800)*

TAB. 4 – Problèmes difficiles d'enchères combinatoires (temps en secondes)

Il faut tout de même rappeler ici combien CPLEX est un logiciel hautement optimisé, de même que son encodage du problème des enchères combinatoires. De notre point de vue, il ne fait pas de doutes que MinSatz puisse être encore optimisé, à la fois pour les problèmes généraux MinSAT, mais aussi pour les problèmes plus spécifiques d'enchères combinatoires. Ainsi, la borne supérieure pourrait être optimisée à l'aide d'une meilleure partition pondérée du graphe ou grâce à des techniques comme la consistance par arcs virtuels [Cooper *et al.* 2010].

Dans la cinquième – et dernière – expérimentation, nous avons étudié l'existence d'éventuelles relations entre la valeur MaxSAT d'une instance aléatoire donnée et sa valeur MinSAT. Ce point est d'ailleurs la raison originelle ayant conduit au travail autour de MinSAT.

Nous avons généré un ensemble de formules 3SAT aléatoires ayant 120 variables au seuil (avec 511 clauses). La valeur MaxSAT de ces instances a permis de les partitionner, chaque partition contenant toutes les formules 3SAT d'une même valeur MaxSAT. La figure 2 montre clairement que pour les 4 partitions obtenues, la courbe CDF (Cumulative Distribution Function) est significativement différente, suggérant que la distribution des valeurs MinSAT est distincte entre partitions. On notera qu'un point (x, y) sur ces courbes signifie que la fraction des instances ayant une valeur MinSAT inférieure à x dans la partition correspondante est y. En d'autres termes, y peut grossièrement se voir comme la probabilité qu'une instance ayant une certaine valeur MaxSAT ait une valeur MinSAT inférieure ou égale à une certaine valeur. De manière à confirmer cette observation, nous avons procédé à des tests de type *two-sample Kolmogorov-Smirnov* entre tous les couples de courbes. L'hypothèse nulle a été rejetée avec une quasi certitude.

Cette dernière expérimentation permet de conclure sur une observation relativement surprenante. En effet, il semble que les instances aléatoires ayant la plus grande valeur MaxSAT sont aussi celles qui ont tendance, de manière significative, à avoir la valeur MinSAT la plus petite. Nous avons aussi pu confirmer ce phénomène avec des ratio r=C/V plus élevés (jusque $r = 8$) et aussi pour un grand nombre d'instances de 80 et 100 variables à différents ratios.

5 Conclusion

Nous avons proposé, dans cet article, et pour la première fois, de voir le problème MinSAT d'un point de vue applicatif et pragmatique. Nous avons proposé le premier solveur MinSAT pondéré partiel, appelé MinSatz. Bien que MinSAT et MaxSAT soient tous deux des extensions naturelles de SAT, l'utilité de MinSAT, en pratique, n'était pas claire avant nos travaux. De plus, bien que les encodages MaxSAT et MinSAT utilisés soient très similaires, il apparaît clairement combien l'utilisation du formalisme MinSAT est extrêmement compétitive par rapport à MaxSAT. Nous pensons que cela est dû au fait que notre solveur permet d'intégrer des mécanismes de calculs de bornes mêlant raisonnement en logique propositionnelle et partitions de graphes, qui sont fondamentalement différents de ceux utilisés par les solveurs MaxSAT, et qui semblent être particulièrement performants pour exploiter les structures de graphe. Nos expérimentations montrent aussi que notre approche est une alternative de choix pour certains problèmes d'optimisations, dans la mesure où les performances affichées dépassent les solveurs dédiés aux problèmes testés (MaxClique et Enchères combinatoires). L'efficacité obtenue nous a aussi permis d'étudier expérimentalement l'existence d'une corrélation possible entre

les valeurs MinSAT et MaxSAT des instances 3SAT aléatoires uniformes.

Dans la suite de ce travail, nous pensons améliorer les techniques de calculs de bornes en utilisant une meilleure partition des graphes considérés, et pousser plus loin l'étude des performances de MinSatz sur les instances d'enchères combinatoires.

Références

[Avidor and Zwick 2005] Adi Avidor and Uri Zwick. Approximating min 2-sat and min 3-sat. *Theory of Computing Systems*, 38(3) :329–345, 2005.

[Cooper *et al.* 2010] M. Cooper, S. de Givry, M. Sanchez, T. Schiex, M. Zytnicki, and T. Werner. Soft arc consistency revisited *Artificial Intelligence*, (7-8) :449-478, 2010

[Fujishima *et al.* 1999] Yuzo Fujishima, Kevin Leyton-Brown, and Yoav Shoham. Taming the computational complexity of combinatorial auctions : Optimal and approximate approaches. In *IJCAI*, pages 548–553, 1999.

[Heras *et al.* 2008] Federico Heras, Javier Larrosa, Simon de Givry, and Thomas Schiex. 2006 and 2007 Max-SAT evaluations : Contributed instances. *JSAT*, 4(2-4) :239–250, 2008.

[Kohli *et al.* 1994] Rajeev Kohli, Ramesh Krishnamurti, and Prakash Mirchandani. The minimum satisfiability problem. *SIAM J. Discrete Mathematics*, 7(2) :275–283, 1994.

[Konc and Janezic 2007] J. Konc and D. Janezic. An improved branch and bound algorithm for the maximum clique problem. *Communications in Mathematical and in Computer Chemistry*, 58 :569–590, 2007.

[Kuegel 2010] Adrian Kuegel. Improved exact solver for the Weighted MAX-SAT problem. In *Workshop Pragmatics of SAT*, 2010.

[Leyton-Brown *et al.* 2000] Kevin Leyton-Brown, Mark Pearson, and Yoav Shoham. Towards a universal test suite for combinatorial auction algorithms. In *ACM Conference on Electronic Commerce*, pages 66–76, 2000.

[Li and Anbulagan 1997] Chu Min Li and Anbulagan, Heuristics Based on Unit Propagation for Satisfiability Problems. In Proceedings of IJCAI, pages 366–371,

[Li and Manyà 2009] Chu Min Li and F. Manyà. MaxSAT, hard and soft constraints. In Armin Biere, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability*, pages 613–631. IOS Press, 2009.

[Li and Quan 2010a] Chu Min Li and Zhe Quan. Combining graph structure exploitation and propositional reasoning for the maximum clique problem. In *ICTAI*, pages 344–351, 2010.

[Li and Quan 2010b] Chu Min Li and Zhe Quan. An efficient branch-and-bound algorithm based on maxsat for the maximum clique problem. In *AAAI*, pages 128–133, 2010.

[Li *et al.* 2007] Chu Min Li, Felip Manyà, and Jordi Planes. New inference rules for Max-SAT. *Journal of Artificial Intelligence Research*, 30 :321–359, 2007.

[Li *et al.* 2010a] Chu Min Li, Felip Manyà, Nouredine Ould Mohamedou, and Jordi Planes. Resolution-based lower bounds in MaxSAT. *Constraints*, 15(4) :456–484, 2010.

[Li *et al.* 2010b] Chu Min Li, Felip Manyà, Zhe Quan, and Zhu Zhu. Exact MinSAT solving. In *SAT*, pages 363–368, 2010.

[Lin *et al.* 2008] Han Lin, Kaile Su and Chu Min Li. Within-Problem Learning for Efficient Lower Bound Computation in Max-SAT Solving. In *AAAI*, pages 351–356, 2008.

[Marathe and Ravi 1996] M. V. Marathe and S. S. Ravi. On approximation algorithms for the minimum satisfiability problem. *Information Processing Letters*, 58 :23–29, 1996.