Un Algorithme de séparation et évaluation Efficace Basé sur MaxSAT pour le Problème Maxclique

Chu Min Li, Zhe Quan

Un Algorithme de séparation et évaluation Efficace Basé sur MaxSAT pour le Problème Maxclique

Chu-Min LI **Zhe Quan**

Université de Picardie Jules Verne,
Amiens, France
chu-min.li@u-picardie.fr quanzhe@gmail.com

Résumé

Fréquemment, Les algorithmes de séparation et évaluation pour le problème Maxclique utilisent une borne supérieure basée sur la partition d'un graphe en ensembles indépendants, pour obtenir la borne supérieure d'une clique maximum du graphe, qui ne peut pas être très fine pour les graphes imparfaits. Dans ce papier, nous proposons un nouveau codage de Maxclique en MaxSAT et l'utilisation de l'approche MaxSAT pour améliorer la borne supérieure basée sur la partition P. De cette manière, la puissance d'algorithmes de partitionnement d'un graphe spécifiques pour Maxclique et la performance de la méthode MaxSAT dans la logique propositionnelle sont naturellement associées pour résoudre Maxclique. Les résultats expérimentaux montrent que l'approche est très efficace sur les graphes aléatoires difficiles et sur les benchmarks Maxclique de DIMACS, et permet de fermer un problème DIMACS ouvert.

Mots-Clés : séparation et évaluation, Maxclique, MaxSAT

Abstract

State-of-the-art branch-and-bound algorithms for the maximum clique problem (Maxclique) frequently use an upper bound based on a partition P of a graph into independent sets for a maximum clique of the graph, which cannot be very tight for imperfect graphs. In this paper we propose a new encoding from Maxclique into MaxSAT and use MaxSAT technology to improve the upper bound based on the partition P. In this way, the strength of specific algorithms for Maxclique in partitioning a graph and the strength of MaxSAT technology in propositional reasoning are naturally combined to solve Maxclique. Experimental results show that the approach is very effective on hard random graphs and on DIMACS Maxclique benchmarks, and allows to close an open DIMACS problem.

Keywords : Branch-and-Bound, Maxclique, MaxSAT

1 Introduction

Considérons un graphe non orienté G=(V, E), où V est un ensemble de n sommets {v_1, v_2, ..., v_n} et E est un ensemble de m arêtes. L'arête (v_i, v_j) avec $i \neq j$ connecte les deux sommets v_i et v_j. Une clique de G est un sous-ensemble C de V tel que tous les deux sommets de C sont connectés par une arête. Le problème Maxclique consiste à trouver une clique de G de cardinalité maximum dénotée par $\omega(G)$.

Un ensemble indépendant de G est un sous-ensemble I de V de sorte que aucun sommet n'est connecté à un autre dans I. Étant donné G, le GCP (Graph Coloring Problem) demande de trouver le nombre minimum de couleurs (i.e., le nombre chromatique $\chi(G)$) nécessaires pour colorer les sommets de G tel que deux sommets connectés ne partagent pas la même couleur. La résolution de GCP est équivalent à partitionner G en un nombre minimum d'ensembles indépendants, parce que tous les sommets colorés par la même couleur dans G constituent un ensemble indépendant. Nous avons $\chi(G) \geq \omega(G)$, puisque chaque sommet dans une clique doit être affecté à une couleur différente. Un graphe G est parfait si $\chi(G')$=$\omega(G')$ pour tout sous-graphe induit G' de G. Maxclique est un très important problème combinatoire NP-difficile, car il apparaît dans beaucoup d'applications du monde réel. De nombreux efforts ont été consacrés à le résoudre. Voir [11] où Pardolos et Xu ont donné un survol des premiers travaux intensifs sur Maxclique. Dans la littérature, nous dis-

tinguons deux types d'algorithmes pour Maxclique : des méthodes d'approximation incluant les algorithmes de recherche locale stochastique, e.g. [12], et les algorithmes exacts incluant les algorithmes par séparation et évaluation, e.g. [14, 5, 10, 1, 2, 13]. Les algorithmes d'approximation sont capables de résoudre des problèmes Maxclique de grande taille, mais ne garantissent pas l'optimalité de leurs solutions. Les algorithmes exacts garantissent l'optimalité des solutions qu'ils trouvent. Dans cet papier, nous nous concentrons sur les algorithmes de séparation et évaluation pour Maxclique.

Les algorithmes de séparation et évaluation pour Maxclique utilisent fréquemment une solution heuristique pour GCP comme une borne supérieure. Il y a deux inconvénients dans cette approche : (i) le nombre d'ensembles indépendants dans une partition calculée de G n'est généralement pas minimum, puisque GCP est lui-même NP-difficile ; (ii) même si la partition est minimale, il peut y avoir une grande différence entre $\chi(G)$ et $\omega(G)$ lorsque G n'est pas parfait, de sorte que $\chi(G)$ n'est pas une borne supérieure de bonne qualité de $\omega(G)$. Ces deux inconvénients pourraient probablement expliquer la stagnation de la recherche sur les algorithmes exacts pour Maxclique. Ainsi, des bornes supérieures de meilleure qualité sont nécessaires dans les algorithmes de séparation et évaluation pour Maxclique.

Récemment des progrès considérables ont été faits dans la résolution de MaxSAT (voir [6] pour une présentation de ces progrès). Etant donné un ensemble de variables booléennes $\{x_1,x_2,...,x_n\}$, un littéral l est une variable x_i ou sa négation $\bar{x}_i$, une clause est un ou logique des littéraux. Une formule CNF ϕ est un ensemble de clauses. Le problème MaxSAT demande de trouver une affectation des valeurs de vérité (0 ou 1) aux variables booléennes pour maximiser le nombre de clauses satisfaites dans ϕ. Maxclique peut être codé en MaxSAT puis résolu en utilisant un solveur MaxSAT. Malheureusement, les solveurs MaxSAT ne sont pas compétitifs pour résoudre Maxclique, parce que leur raisonnement n'est pas guidé par les propriétés structurelles du graphe.

Dans cet papier, nous montrons que la technologie développée pour résoudre MaxSAT peut être utilisée pour améliorer les bornes supérieures dans l'algorithme de séparation et évaluation pour Maxclique. Nous présentons tout d'abord quelques préliminaires et un algorithme de séparation et évaluation de base pour Maxclique appelé MaxCLQ, avant de présenter quelques bornes supérieures précédentes pour Maxclique. Ensuite, nous proposons un nouveau codage de Maxclique en MaxSAT, ce qui nous permet d'utiliser la technologie MaxSAT pour améliorer les précédente bornes supérieures pour Maxclique. Nous étudions ensuite le comportement de notre approche intégrée dans MaxCLQ, et comparons MaxCLQ avec les meilleurs algorithmes exacts à nos connaissances sur les graphes aléatoires et sur les benchmarks Maxclique de DIMACS qui est largement utilisés [1] pour évaluer les algorithmes de Maxclique dans la littérature. Les résultats expérimentaux montrent que MaxCLQ est très efficace grâce à la technologie MaxSAT intégrée puisqu'il est, à notre connaissance, capable de résoudre un problème ouvert DIMACS pour la première fois.

2 Preliminaires

Une clique C d'un graphe G=(V, E) est maximum si aucune clique de cardinalité plus grande n'existe dans G. G peut avoir plusieurs cliques maximums. Soit V' un sous-ensemble de V, le sous-graphe G' de G induit par V' est défini comme G'=(V', E'), où E' = $\{(v_i, v_j) \in E \mid v_i, v_j \in V'\}$. Etant donné un sommet v de G, l'ensemble des sommets voisins de v est noté $\Gamma(v)$=$\{v' \mid (v, v') \in E\}$. La cardinalité $|\Gamma(v)|$ de $\Gamma(v)$ est appelée le degré de v. G_v désigne le sous-graphe induit par $\Gamma(v)$ et $G\backslash v$ est le graphe induit par $V\backslash\{v\}$. $G\backslash v$ est obtenue en enlevant v et toutes les arêtes connectant v à G. La densité d'un graphe de n sommets et m arêtes est égale à $2m/(n(n-1))$.

Soit un problème MaxSAT ϕ, une affectation de valeurs de vérité aux variables boolennes satisfait un littéral positif x si x=1, satisfait un littéral négatif $\bar{x}$ si x=0, et satisfait une clause si au moins un littéral dans la clause est satisfait. Nous distinguons spécialement les clauses unitaires qui ne contiennent qu'un seul littéral, et les clauses vides qui ne contiennent pas de littéral et ne peuvent pas être satisfaites. Un problème MaxSAT ϕ est dit partiel, si certaines clauses de ϕ sont dures, c'est-à-dire qu'elles doivent être satisfaites dans toutes les solutions, et les autres clauses sont souples et peuvent être violées dans une solution. Un problème MaxSAT partiel demande de trouver une affectation satisfaisant toutes les clauses dures en maximisant le nombre de clauses souples satisfaites.

La façon habituelle de coder un problème Maxclique en un problème MaxSAT est d'introduire une variable booléenne x pour chaque sommet v de G, avec le sens que x=1 si et seulement si v appartient à la clique maximum cherchée. Ensuite, une clause binaire dure $\bar{x}_i \vee \bar{x}_j$ est ajoutée pour chaque paire de sommets v_i et v_j qui ne sont pas connectés, signifiant que v_i et v_j ne peuvent pas être dans la même clique. Il est clair que toute affectation satisfaisant toutes les clauses dures donne une clique. Enfin, une clause unitaire souple x est ajoutée pour chaque sommet v. Maximiser le nombre de clauses souples satisfaites, tout en satisfaisant toutes les clauses dures donne une clique maximum.

Par exemple, l'instance Maxclique dans la figure 1 peut être codée en une instance MaxSAT composée des clauses souples : $\{x_1, x_2, x_3, x_4, x_5, x_6\}$, et des clauses dures :

1. disponibles à http://cs.hbg.psu.edu/ txn131/clique.html

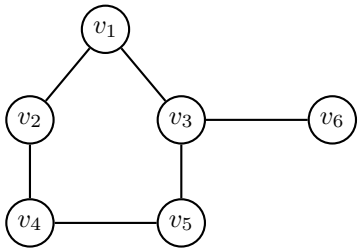

FIGURE 1 – Un simple graphe imparfait ($\chi(G)$=3 et $\omega(G)$=2)

$\{\bar{x}_1 \vee \bar{x}_4, \bar{x}_1 \vee \bar{x}_5, \bar{x}_2 \vee \bar{x}_3, \bar{x}_2 \vee \bar{x}_5, \bar{x}_3 \vee \bar{x}_4, \bar{x}_1 \vee \bar{x}_6, \bar{x}_2 \vee \bar{x}_6, \bar{x}_4 \vee \bar{x}_6, \bar{x}_5 \vee \bar{x}_6\}$.

3 Un Algorithme de séparation et évaluation de base pour Maxclique

L'Algorithme 1 montre le pseudo-code d'un algorithme de séparation et évaluation pour Maxclique, inspiré par l'algorithme de séparation et évaluation MaxSatz pour MaxSAT [7]. Nous illustrons le principe de cet algorithme en utilisant le graphe G de la figure 1. Initialement C=$\emptyset$ et LB=0, et MaxCLQ (G, $\emptyset$, 0) trouve une clique maximum ($\{v_3, v_6\}$) de G comme suit.

Algorithm 1: MaxCLQ(G, C, LB), un algorithme de séparation et évaluation de base pour Maxclique

Input: Un graphe G=(V, E), une clique C, et une borne inférieure LB représentant la cardinalité de la plus grande clique trouvée jusqu'à présent
Output: $C \cup C'$, où C' est une clique maximum de G, si $|C \cup C'| > LB$, $\emptyset$ sinon

```
1  begin
2      si |V|=0 alors retourner C ;
3      UB ← surestimation(G)+|C| ;
4      si LB ≥ UB alors retourner ∅ ;
5      choisir un sommet v de degré minimum de G ;
6      C1 ← MaxCLQ(Gv, C ∪{v}, LB) ;
7      si |C1| > LB alors LB ← |C1| ;
8      C2 ← MaxCLQ(G\v, C, LB) ;
9      si |C1| ≥ |C2| alors retourner C1 ; sinon
       retourner C2 ;
10 end
```

Dans la ligne 3, la function surestimation(G) donne une borne supérieure clairement supérieure à 0 pour une clique maximum de G, puisque G n'est pas vide. Ensuite MaxCLQ choisit v_6 comme sommet de branchement (ligne 5), car il est de degré minimum. Ainsi toutes les cliques de G sont implicitement divisées en deux ensembles : l'ensemble des cliques contenant v_6 et l'ensemble des cliques ne contenant pas de v_6. Le premier appel récursif (ligne 6) MaxCLQ(G_{v_6}, $\{v_6\}$, 0) retourne une clique ($\{v_3, v_6\}$) contenant v_6, où G_{v_6}=($\{v_3\}$, $\emptyset$).

Puis, la ligne 7 change la borne inférieure LB à 2. Le deuxième appel récursif MaxCLQ($G\backslash v_6$, $\emptyset$, 2) essaie de trouver une clique maximum ne contenant pas de v_6 et plus grande que 2 dans $G\backslash v_6$ (ligne 8), où $G\backslash v_6$ est le cycle composé de $\{v_1,v_2,v_3,v_4,v_5\}$. Si la fonction surestimation($G\backslash v_6$) utilise une méthode de coloration et retourne 3 comme borne supérieure (rappelons que $\chi(G\backslash v_6)$=3) pour une clique maximum de $G\backslash v_6$, MaxCLQ($G\backslash v_6$, $\emptyset$, 2) doit faire deux autres appels récursifs en choisissant un sommet de branchement, e.g. v_1 : MaxCLQ(($G\backslash v_6$)$_{v_1}$, $\{v_1\}$, 2) et MaxCLQ(($G\backslash v_6$)$\backslash v_1$, $\emptyset$, 2). L'exécution de ces deux appels retourne tous les deux $\emptyset$. Ainsi, MaxCLQ($G\backslash v_6$, $\emptyset$, 2) retourne $\emptyset$ à la ligne 9. Cependant, si la fonction surestimation($G\backslash v_6$) utilise l'approche proposée dans ce papier, il retournera 2 comme borne supérieure, de telle sorte que UB=LB et MaxCLQ($G\backslash v_6$, $\emptyset$, 2) retourne directement $\emptyset$ (line 4) sans d'autres appels récursifs, puisque UB=LB signifie qu'une clique de cardinalité plus grande ne peut pas être trouvée.

Enfin, MaxCLQ(G, $\emptyset$, 0) retourne C_1=$\{v_3, v_6\}$ à la ligne 9.

4 Survol des bornes supérieures précédentes pour Maxclique

Les algorithmes de séparation et évaluation pour Maxclique utilisent fréquemment des solutions heuristiques de GCP qui peuvent être obtenues en un temps raisonnable comme leurs bornes supérieures. Cette approche se base sur la propriété suivante :

Proposition 1 *Soit $\omega(G)$ la cardinalité d'une clique maximum du graphe G. Si G peut être partitionné en k ensembles indépendants, alors $\omega(G) \leq k$.*

Comme un prétraitement, l'algorithme Cliquer [10] partitionne G, en déterminant un ensemble indépendant à la fois. Tant qu'il y a des sommets qui peuvent être ajoutés dans l'ensemble indépendant en construction, l'un de ces sommets avec le plus grand degré est ajouté. Puis Cliquer construit une clique maximum incrémentalement en considérant les sommets dans l'ordre inverse de celui dans lequel les sommets sont ajoutés dans les ensembles indépendants, la borne supérieure des cliques maximums contenant un sommet donné est rapidement déterminée de cette façon.

Falhe [2] améliore l'algorithme de Carraghan et Pardalos [1], en utilisant l'heuristique constructive DSATUR pour colorer les sommets un par un, tout en partitionnant en parallèle le graphe en ensembles indépendants. Comme les sommets sont colorés ou insérés dans un ensemble indépendant dans l'ordre décroissant ou croissant selon leur degré, quatre solutions heuristiques de GCP sont obtenues, et la meilleure est utilisée comme la borne supérieure.

Regin [13] utilise une borne supérieure basée sur un algorithme de matching. Un matching, qui correspond à un ensemble d'ensembles indépendants de taille 2 ici, est calculé en parcourant les sommets d'un graphe, et en considérant qu'une arête existe si deux sommets ne sont pas connectés.

MCQ [15] colorie les sommets dans un ordre prédéterminé. Supposons que les ensembles indépendants actuels sont S_1, S_2, ..., S_k (dans cet ordre, k est égal à 0 au début du processus de coloration), MCQ insère le premier sommet v dans l'ordre prédéterminé dans le premier S_i tel que v est non-connecté à tous les sommets déjà dans S_i. Si une telle S_i n'existe pas, une nouvel ensemble indépendant S_{k+1}) est ouvert et v est inséré dedans. Après que tous les sommets sont insérés dans les ensembles indépendants, ils sont réordonnés en fonction de leur ensemble indépendant, les sommets de S_i précédant les sommets de S_j si $i < j$. Ce procédé de coloration est exécuté pour G_v après chaque branchement sur le sommet v. L'ordre prédéterminé de sommets G_v est hérité de G. MCR [14] améliore MCQ avec un meilleur ordre de sommets du graphe d'entrée initial, mais utilise le même processus de coloration pour calculer la borne supérieure.

MaxCliqueDyn [5] est également obtenu en améliorant MCQ. Alors que MCQ (ainsi que MCR) ne calcule que le degré des sommets à la racine de l'arbre de recherche pour le graphe d'entrée initial, MaxCliqueDyn recalcule dynamiquement le degré des sommets pour certains nœuds situés à proximité de la racine de l'arbre de recherche choisis en utilisant un paramètre, et réordonne les sommets dans l'ordre décroissant de leur degré avant la coloration de ces sommets. Le calcul dynamique des degrés près de la racine de l'arbre de recherche rend MaxCliqueDyn plus rapide que MCQ pour les graphes aléatoires lorsque leur densité est comprise entre 0,7 à 0,95, mais plus lent que MCQ lorsque la densité des graphes est inférieure à 0,7.

5 Nouveaux codages de Maxclique en MaxSAT

Definition 1 *Soit G un graphe partitionné en ensembles indépendants, le codage de Maxclique en MaxSAT basé sur les ensembles indépendants est défini comme suit : (1) chaque sommet v de G est représenté par une variable booléenne x, (2) une clause dure $\bar{x}_i \vee \bar{x}_j$ est ajoutée pour chaque paire de sommets non-connectés (v_i, v_j), et (3) une clause souple est ajoutée pour chaque ensemble indépendant qui est un ou logique des variables représentant les sommets dans l'ensemble indépendant.*

Il est facile de voir que le codage habituel Maxclique en MaxSAT présenté dans la section préliminaire est un cas particulier de la définition 1, dans lequel chaque ensemble indépendant ne contient qu'un seul sommet.

Par exemple, le graphe de la figure 1 peut être partitionné en trois ensembles indépendants $\{v_1, v_4, v_6\}$, $\{v_2, v_3\}$, $\{v_5\}$. Par conséquent, le codage de Maxclique en MaxSAT basé sur les ensembles indépendants contient trois clauses souples : $\{x_1 \vee x_4 \vee x_6, x_2 \vee x_3, x_5\}$, et les mêmes clauses dures comme dans le codage habituel : $\{\bar{x}_1 \vee \bar{x}_4, \bar{x}_1 \vee \bar{x}_5, \bar{x}_2 \vee \bar{x}_3, \bar{x}_2 \vee \bar{x}_5, \bar{x}_3 \vee \bar{x}_4, \bar{x}_1 \vee \bar{x}_6, \bar{x}_2 \vee \bar{x}_6, \bar{x}_4 \vee \bar{x}_6, \bar{x}_5 \vee \bar{x}_6\}$.

On observe que les clauses dures exigent que au plus un littéral puisse être satisfait dans une clause souple. Etant donné une affectation satisfaisant toutes les clauses dures, une clause souple satisfaite a exactement un littéral satisfait du fait des clauses dures. L'ensemble indépendant correspondant a un sommet dans la clique donnée par l'affectation. Donc, nous avons :

Proposition 2 *Soit ϕ une instance MaxSAT partielle codant une instance Maxclique en basant sur une partition du graphe G en ensembles indépendants, l'ensemble des variables booléennes de ϕ affectées à vrai dans une solution optimale de ϕ donne une clique maximum de G.*

Différentes partitions de G en ensembles indépendants donnent différentes instances MaxSAT. Le tableau 1 compare trois partitions des graphes aléatoires de 200 sommets et de certains graphes DIMACS. Le codage Max est basé sur une partition calculée en utilisant l'algorithme de coloration MCQ pour insérer des sommets un par un dans un ensemble indépendant dans l'ordre décroissant de leur degré (i.e., les sommets avec le degré maximum sont insérés en premier). Min est similaire à Max sauf que les sommets sont insérés dans l'ordre croissant de leur degré (i.e., les sommets avec le degré minimum sont insérés en premier). Le codage Habituel correspond à celui présenté dans la section préliminaire, dans lequel chaque clause souple est unitaire. Nous reportons, pour chaque codage, le nombre d'ensembles indépendants dans le graphe, et le temps nécessaire pour trouver une clique maximum par deux solveurs MaxSAT de l'état de l'art : MaxSatz [7] et Minimaxsat [4]. A chaque densité des graphes aléatoires, 50 graphes sont générés et codés en MaxSAT. Le runtime moyen et le nombre moyen d'ensembles indépendants k sont reportés, k étant aussi le nombre de clauses souples dans le codage. Pour les deux solveurs MaxSAT, le codage Max est le meilleur, le nombre de clauses souples étant le plus petit. Nous allons utiliser ce codage pour appliquer la technologie MaxSAT dans la section suivante.

Heras et Larrosa [3] proposent un prétraitement pour améliorer le codage habituel en utilisant deux règles d'inférence basées sur la résolution MaxSAT. Ce prétraitement permet d'améliorer la borne supérieure initiale basée sur le nombre de clauses souples initiales dans le codage habituel, mais ne semble pas améliorer le temps de Mini-MaxSAT pour résoudre les instances DIMACS. Toutefois, le prétraitement permet d'accélérer Minimaxsat pour certaines instances spécifiques avec solution optimale cachée

TABLE 1 – Durée [sec.] de Maxsatz (version 2009, MSZ dans le tableau) et Minimaxsat (Mini dans le tableau) sur un MacPro 2,8 GHz avec 4 Go de mémoire pour trois codages différents de Maxclique en MaxSAT, k est le nombre de clauses souples dans un codage

$Graph$		$Habituel$			Min			Max		
name	density	k	MSZ	Mini	k	MSZ	Mini	k	MSZ	Mini
200	0.40	200	2.11	2.25	30.6	1.31	0.80	27.3	1.10	**0.76**
200	0.60	200	36.8	18.3	45.4	11.2	7.68	40.8	**7.22**	7.27
200	0.80	200	9691	576.3	67.4	265.7	278.5	61.1	**117.8**	269.2
brock200_1	0.74	200	1344	156.7	60	103.6	100.4	56	**52.9**	67.5
brock200_2	0.50	200	6.72	4.36	36	2.58	2.09	33	2.28	**1.79**
brock200_3	0.61	200	45.3	13.7	45	9.44	7.95	43	7.48	**6.88**
keller4	0.65	171	40.5	34.0	34	6.16	8.81	32	**2.89**	7.39
p_hat300-1	0.24	300	3.09	2.67	33	2.51	1.12	22	1.81	**0.96**
p_hat300-2	0.49	300	105.3	7.54	69	69.7	3.64	46	3.06	**2.72**
p_hat300-3	0.74	300	>3h	452.1	95	2326	163.5	72	185.3	**129.2**

et certaines protein alignment instances.

Malgré que les deux nouveaux codages permettent un gain significatif pour MaxSatz et MiniMaxSAT par rapport au codage habituel, les solveurs MaxSAT ne sont pas encore en mesure de battre les solveurs Maxclique spécifiques sur les instances aléatoires et les instances DIMACS, comme nous allons voir dans la section 7. Nous pensons que la raison principale de cette inefficacité des solveurs MaxSAT est que ces solveurs n'exploitent pas la structure des graphes, contrairement aux algorithmes spécifiques. En revanche, nous allons montrer que l'association de l'exploitation de structure des graphes et la puissance de raisonnement propositionnel de la technologie MaxSAT permet grandement d'ameliorer les algorithmes spécifiques pour Maxclique.

6 Utilisation de la technologie MaxSAT pour améliorer la borne supérieure de Maxclique

Etant donnée une instance MaxSAT partiel, un solveur MaxSAT de séparation et évaluation doit trouver une affectation qui minimise le nombre de clauses souples insatisfaites et satisfait toutes les clauses dures. Dans ce but, le solveur doit sous-estimer, à chaque noeud de l'arbre de recherche, le nombre de clauses souples qui seront violées par une affectation complète étendant l'affectation partielle en cours. Une approche proposée dans [9] et qui s'est révélée très puissante dans MaxSatz et Minimaxsat consiste à détecter des sous-ensembles inconsistants de clauses souples disjoints. Un sous-ensemble de clauses souples est inconsistant si le sous-ensemble, en conjonction avec l'ensemble des clauses dures, permet de déduire une contradiction (une clause vide). Le nombre de sous-ensembles inconsistants de clauses souples disjoints est une minoration du nombre de clauses souples violées par une affectation complète étendant l'affectation partielle en cours et satisfaisant toutes les clauses dures, car il y a au moins une clause souple violée dans chaque sous-ensemble inconsistant de clauses souples.

Cette approche, appelée UP, détecte les sous-ensembles inconsistants de clauses disjoints avec la propagation des clauses unitaires comme suit :

Etant donnée une formule CNF ϕ, UP stocke toutes les clauses unitaires de ϕ dans une file Q, et applique la propagation des clauses unitaires dans une copie φ de ϕ, en successivement prenant une clause unitaire l de Q pour la satisfaire (i.e., l=1). La satisfaction de l signifie que toutes les clauses contenant l sont satisfaites et retirées de φ, et que $\bar{l}$ est enlevé des autres clauses, ce qui peut produire des nouvelles clauses unitaires et des clauses vides dans φ (une clause devient vide si elle ne contenait que $\bar{l}$). Les nouvelle clauses unitaires sont stockées à la fin de Q. La propagation des clauses unitaires continue jusqu'à ce que'il n'existe plus de clause unitaire dans Q ou jusqu'à ce qu'une clause vide soit produite. Dans ce dernier cas, les clauses utilisées dans la propagation pour produire la clause vide constituent un sous-ensemble inconsistant de clauses. Une fois qu'un sous-ensemble inconsistant des clauses est identifié, ces clauses sont retirées de ϕ. Ainsi UP continue de détecter d'autres sous-ensembles inconsistants de clauses tant qu'il reste des clauses unitaires dans ϕ. UP est amélioré dans [8] par la détection des littéraux d'échec. Un littéral l est d'échec si, quand il est satisfait et $\bar{l}$ retiré de toutes les clauses le contenant, la propagation des clauses unitaires produit une clause vide. Si les deux littéraux l et $\bar{l}$ sont d'échec, l'union des clauses utilisées pour produire les deux clauses vides constitue un sous-ensemble inconsistant.

Dans le cas de MaxSAT partiel, après l'obtention d'un sous-ensemble inconsistant des clauses en utilisant les approches ci-dessus, les clauses dures dans le sous-ensemble ne sont pas retirées de ϕ et peuvent être re-utilisées dans la détection d'autres contradictions, mais les clauses souples dans le sous-ensemble sont retirées de ϕ. En d'autres termes, les clauses dures peuvent appartenir à plusieurs sous-ensembles inconsistants de clauses, parce qu'elles doivent être satisfaites de toute façon. Par contre, une clause souple ne peut pas appartenir à deux sous-ensembles inconsistants de clauses différents.

Nous adaptons la détection des littéraux d'échec pour détecter des sous-ensembles inconsistants de clauses souples dans une instance MaxSAT partielle codant une instance MaxClique. Nous ne détectons pas si un littéral négatif est d'échec ou non, car une variable peut avoir plusieurs occurrences négatives, mais une seule occurrence positive (dans une clause souple). Au lieu de cela, nous détectons si chaque littéral dans une clause souple est d'échec. En fait, soit une clause souple $c = x_1 \vee x_2 \vee ... \vee x_t$, si tous les x_i ($1 \leq i \leq t$) sont littéral d'échec, l'union de toutes les clauses souples utilisées pour produire une clause vide pour chaque x_i, avec c, constitue un sous-ensemble inconsistant.

Par exemple, dans l'instance MaxSAT codant le graphe de la Figure 1 basée sur les ensembles indépendants présentée dans la dernière section, il y a trois clauses souples $\{x_1 \vee x_4 \vee x_6,\ x_2 \vee x_3,\ x_5\}$ correspondant aux trois ensembles indépendants dans une partition optimale du graphe. Les clauses dures sont : $\{\bar{x}_1 \vee \bar{x}_4, \bar{x}_1 \vee \bar{x}_5, \bar{x}_2 \vee \bar{x}_3, \bar{x}_2 \vee \bar{x}_5, \bar{x}_3 \vee \bar{x}_4, \bar{x}_1 \vee \bar{x}_6, \bar{x}_2 \vee \bar{x}_6, \bar{x}_4 \vee \bar{x}_6, \bar{x}_5 \vee \bar{x}_6\}$. La borne supérieure initiale d'une clique maximum est 3, qui est la meilleure borne supérieure qui peut être obtenue en utilisant un procédé de coloration. Nous montrons que x_5 est un littéral d'échec, permettant d'améliorer la borne supérieure. Posons x_5=1 pour satisfaire la troisième clause souple, $\bar{x}_5$ est retiré des clauses dures $\bar{x}_1 \vee \bar{x}_5$, $\bar{x}_2 \vee \bar{x}_5$, et $\bar{x}_5 \vee \bar{x}_6$, les trois nouvelles clauses unitaires impliquent que x_1=0, x_2=0 et x_6=0, puis x_1 et x_6 soient retirés de la première clause souple, et x_2 de la deuxième, les deux clauses devenant unitaires. Donc, x_4 et x_3 doivent être affectés à 1, rendant la clause dure $\bar{x}_3 \vee \bar{x}_4$ vide. Donc, x_5 est un littéral d'échec et les trois clauses souples utilisées dans la propagation pour produire la clause vide constituent un sous-ensemble inconsistant, parce que ce sous-ensemble, en conjonction avec les clauses dures, permet de déduire une contradiction. Puisqu'au plus deux clauses souples peuvent être satisfaites par une affectation satisfaisant toutes les clauses dures, nous améliorons la borne supérieure pour une clique maximum de 3 à 2.

En général, nous avons :

Proposition 3 *Soit $\omega(G)$ la cardinalité d'une clique maximum d'un graphe G. Si G peut être partitionné en k ensembles indépendants, et il existe s sous-ensembles inconsistants de clauses souples disjoints dans l'instance MaxSAT codant G en basant sur la partition de G en k ensembles indépendants, alors $\omega(G) \leq k - s$.*

En se basant sur la proposition 3, la fonction surestimation(G) présentée dans l'algorithme 2 partitionne d'abord G en ensembles indépendants de la même manière que MCQ, sauf que le degré de chaque sommet de G est exact. Par conséquent, la qualité de la partition est sans doute meilleure (i.e., la partition contient vraisemblablement moins d'ensembles indépendants), puisque les sommets plus contraints (i.e. avec plus de voisins connectés) sont insérés en premier dans les ensembles indépendants. Ensuite la fonction code le graphe en une instance MaxSAT basée sur la partition, ce qui correspond au codage Max présenté dans la dernière section, et utilise la technologie MaxSAT efficace pour améliorer la borne supérieure donnée par la partition.

Avec l'utilisation de la fonction surestimation(G), la différence essentielle entre MaxCLQ et un solveur MaxSAT pour Maxclique se présente comme suit. Alors qu'un solveur MaxSAT utilise un codage MaxSAT fixe et ne repartitionne jamais de sous-graphe pendant la recherche, MaxCLQ re-partitionne de façon dynamique le sous-graphe à chaque noeud de l'arbre de recherche et code ce sous-graphe en MaxSAT pour améliorer la borne supérieure donnée par la partition, de sorte que la force des algorithmes spécifiques pour Maxclique dans le partitionnement de graphe et la puissance de la technologie MaxSAT dans le raisonnement propositionnel sont naturellement combinées dans MaxCLQ pour résoudre Maxclique.

Algorithm 2: surestimation(G), une surestimation de la cardinalité d'une clique maximum de G

Input: Un graphe G=(V, E)
Output: Borne supérieure pour une clique maximum de G

```
1  begin
2      P ← ∅;
3      G' ← G;
4      while G' est non vide do
5          v ← le sommet de dégrée maximum de G';
6          retirer v de G';
7          if il exist un ensemble indépendant S dans P
           où v est non-connecté à tous les sommets then
8              insérer v dans S ;
9          else
10             créer un nouvel ensemble indépendant S;
11             insérer v dans S;
12             P ← P ∪ {S} ;
13     Coder G en une instance MaxSAT φ en basant sur
       P;
14     s ← 0;
15     while φ contient une clause souple qui n'est pas
       testée do
16         c ← la clause souple de φ de taille minimum
           qui n'est pas testée;
17         marquer c comme "testée";
18         if chaque littéral dans c est d'échec then
19             retirer c et toutes les clauses souples
               faisant les littéraux de c d'échec de φ;
20             s ← s+1 ;
21     retourner |P| − s ;
22 end
```

7 Evaluation comparative de MaxCLQ

Nous comparons la performance de MaxCLQ avec celle de MaxCLQ$^-$ qui est identique à MaxCLQ sauf qu'il n'utilise pas la technologie MaxSAT pour améliorer la borne supérieure dans la fonction surestimation(G) (i.e. la fonction retourne toujours |P| dans MaxCLQ$^-$). Nous comparons aussi MaxCLQ avec les meilleurs algorithmes exacts pour Maxclique à notre connaissance : Cliquer, Algorithme Regin, MCR, MaxCliqueDyn (MCQdyn en bref). Le calcul de

la borne supérieure dans MaxCLQ$^-$ est le même que MCR et MCQDyn, à l'exception que MCR calcule seulement le degré de chaque sommet une fois dans la racine de l'arbre de recherche, et MCQdyn recalcule en plus le degré de chaque sommet dans certains noeuds proches de la racine choisis en utilisant un parametre, alors que MaxCLQ$^-$ recalcule le degré de chaque sommet dans tous les noeuds de l'arbre de recherche. En utilisant des informations exactes de degré de sommets, la fonction surestimation(G) donne à priori une borne supérieure de meilleure qualité dans MaxCLQ$^-$ que dans MCR et MCQdyn, mais le recalcul de degré de chaque sommet dans chaque noeud de l'arbre de recherche est consommateur du temps, surtout quand l'arbre de recherche est grand.

Les exécutions de MaxCLQ, MaxCLQ$^-$, Cliquer et MCQdyn sont réalisées sur un MacPro Xeon 2,8 GHz avec processeur Intel et 4 Go de mémoire (produit au début de 2008) avec MAC OS X 10.5. Nous utilisons la dernière version de Cliquer publiée en 2008 [2] et nous l'exécutons avec ses paramètres par défaut. MCQdyn a été obtenu de l'un de ses auteurs (D. Janezic) en janvier 2010 et exécuté avec le meilleur paramètre 0,025. Les temps d'exécutions de MCR et Regin sont normalisés à partir de ceux reportés comme suit.

Les temps d'exécution du programme benchmark dfmax pour les graphes DIMACS r100.5, r200.5, r300.5, r400.5 et r500.5 sur le MacPro sont respectivement de 0,002, 0,033, 0,270, 1,639, 6,281. Le temps d'exécution de dfmax reporté pour l'ordinateur (Pentium4 2,20 GHz CPU avec Linux) exécutant MCR est respectivement 0,00213, 0,0635, 0,562, 3,48, 13,3 pour les mêmes graphes. Ainsi, nous divisons les temps d'exécution de MCR par 2,12 (= (13.3/6.281 +3,48/1,639)/2, la moyenne des deux plus grands ratios). Les temps d'exécution de dfmax pour r100.5-r500.5 de l'ordinateur Regin (Pentium4 2Ghz mobile avec 512 Mo de mémoire) ne sont pas reportés, mais l'ordinateur Regin est environ 10% plus lent que l'ordinateur qui exécute MCR, donc nous divisons les temps d'exécution reportés de Regin par 2,33. Cette normalisation est basée sur la méthode établie dans le Second Challenge d'Implémentation DIMACS pour Cliques, Coloration, et Satisfiabilité, elle est également utilisée dans [14] pour comparer MCR avec d'autres algorithmes. Regin [13] normalise les temps d'exécution des algorithmes en comparant les différents ordinateurs. Parmi les nombreux algorithms comparés avec MCR dans [14] et avec Regin dans [13], nous citons l'algorithme Fahle [2] qui améliore l'algorithme proposé par Carraghan et Pardalos [1] et implémenté dans dfmax. MCR et Regin sont significativement plus rapide que l'algorithme de Fahle.

Le tableau 2 compare les temps d'exécution réels de MaxCLQ, MaxCLQ$^-$, Cliquer et MCQdyn avec les temps d'exécution normalisés de MCR pour les graphes aléatoires jusqu'à 500 sommets. MaxCLQ, MaxCLQ$^-$, Cliquer et MCQdyn résolvent 50 graphes à chaque point et le temps d'exécution moyen est reporté. Les graphes de faible densité (par exemple, des graphes de 150 sommets et de densité 0,7) qui sont résolus en moins de 1 seconde par l'ensemble des cinq algorithmes sont exclus pour économiser l'espace. MaxCLQ est sensiblement meilleur que MaxCLQ$^-$ et l'accélération augmente avec la densité lorsque le nombre de sommets est fixe, et avec le nombre de sommets lorsque la densité est fixe. MCR est plus rapide que MaxCLQ pour les graphes de relativement faible densité (densité $< 0,7$), car il n'a pas à recalculer le degré des sommets à chaque noeud de l'arbre de recherche sauf pour celui de la racine. Toutefois, de façon substantielle MaxCLQ est plus rapide que MCR et que d'autres algorithmes pour les graphes de densité $\geq 0,7$, et l'accélération croit aussi avec le nombre de sommets et la densité du graphe.

TABLE 2 – Temps d'exécution [s] pour les graphes aléatoires. Pour Cliquer, MCQdyn et MaxCLQ$^-$, "-"signifie que l'instance ne peut être résolue en moins de 3 heures ; pour les MCR, "-" signifie que le temp d'exécution n'est pas disponible. Les temps d'exécution de Regin pour les graphes aléatoires ne sont pas disponibles. s est l'amélioration moyenne de la borne supérieure par MaxSAT dans un noeud d'arbre, et Rate est le taux de réussite de la technologie MaxSAT dans MaxCLQ pour couper les sous-arbres (expliqué plus loin), en moyenne de 50 graphes à chaque point.

n	density	Cliquer	MCR	MCQdyn	MaxCLQ$^-$	MaxCLQ	s	Rate
150	0.80	3.49	0.36	0.32	0.64	**0.16**	2.85	0.85
150	0.90	433.3	3.59	1.74	2.69	**0.25**	4.15	0.87
150	0.95	1513	2.01	0.74	1.17	**0.05**	2.68	0.54
200	0.70	2.06	**0.44**	0.47	1.06	**0.44**	2.35	0.82
200	0.80	125.0	8.32	5.12	10.12	**2.27**	3.19	0.87
200	0.90	-	462.4	90.73	138.3	**9.98**	4.87	0.91
200	0.95	-	-	81.4	121.9	**2.40**	6.10	0.90
300	0.60	3.27	**0.87**	1.02	2.32	1.49	2.18	0.74
300	0.70	112.1	14.57	12.12	28.25	**10.29**	2.81	0.82
300	0.80	-	844.3	423.8	805.9	**158.3**	3.39	0.88
300	0.90	-	-	-	-	**6695**	5.52	0.92
500	0.50	7.08	**2.13**	2.87	6.87	6.25	2.17	0.60
500	0.60	176.7	**37.71**	38.84	107.9	54.27	2.70	0.74
500	0.70	-	1797	1496	3527	**1147**	2.95	0.83

Le tableau 3 compare les temps d'exécution réels de MaxCLQ, MaxCLQ$^-$, Cliquer et MCQdyn avec les temps d'exécution normalisés de MCR et Regin sur les benchmarks Maxclique DIMACS. Afin d'économiser de l'espace, nous excluons les instances très faciles qui sont résolues par les six algorithmes en moins de 2 secondes et les cinq instances ouvertes (MANN_a81, hamming10-4, johnson32-2-4, keller6, et p_hat1500-3) qu'aucun algorithme exact à nos connaissances n'est capable de résoudre. Sauf 8 instances faciles que MaxCLQ résout aussi rapidement, MaxCLQ est nettement plus rapide que tous les autres algorithmes, surtout pour les graphes durs et denses. En particulier, MaxCLQ est capable de fermer l'instance ouverte p_hat1000-3.

Dans un arbre de recherche de MaxCLQ, soit p le nombre de noeuds dans lesquels la partition donne déjà

2. disponible à http ://users.tkk.fi/pat/cliquer.html

TABLE 3 – Temps d'exécution [sec] pour DIMACS benchmarks. "d" correspond à la densité. Pour Cliquer, MCQdyn et MaxCLQ$^-$, "-" signifie que l'instance ne peut pas être résolue en moins de 24 heures, sauf dans le cas de keller5, p_hat1500-2 et p_hat1000-3 qui ne peut pas être résolu en moins de 5 jours ; pour Regin et MCR, "-" signifie que le temps d'exécution n'est pas disponible. s est l'amélioration moyenne de la borne supérieure par MaxSAT dans un noeud d'arbre, et Rate est le taux de réussite de la technologie MaxSAT dans MaxCLQ pour couper les sous-arbres (expliqué plus loin).

name	n	d	ω	Cliquer	Regin	MCR	MCQdyn	MaxCLQ$^-$	MaxCLQ	s	Rate
brock200_1	200	0.74	21	6.37	4.60	1.13	0.96	2.28	**0.67**	2.66	0.86
brock400_1	400	0.75	27	22182	4867	1137	703.5	1447	**370.84**	2.92	0.86
brock400_2	400	0.75	29	5617	3395	465.10	309.0	664.9	**178.70**	2.86	0.86
brock400_3	400	0.75	31	1667	1922	766.51	565.0	971.3	**290.06**	2.81	0.85
brock400_4	400	0.75	33	247.7	2597	409.43	320.4	605.6	**167.30**	3.15	0.85
brock800_1	800	0.65	23	-	-	10712	8821	22821	**8815**	2.92	0.80
brock800_2	800	0.65	24	-	-	9679	8125	21001	**7690**	2.77	0.81
brock800_3	800	0.65	25	26014	-	6546	5565	13559	**5285**	2.73	0.80
brock800_4	800	0.65	26	6108	-	4561	4240	9625	**3880**	2.71	0.80
MANN_a27	378	0.99	126	-	7.93	1.98	3.10	6.86	**0.66**	3.21	0.75
MANN_a45	1035	0.996	345	-	-	2931	2006	8965	**255.67**	6.30	0.91
hamming10-2	1024	0.99	512	0.19	0.45	**0.16**	2.26	69.54	7.92	0.34	0.07
keller5	776	0.75	27	-	-	-	31038	78505	**9687**	3.60	0.87
p_hat300-3	300	0.74	36	496.6	17.47	7.45	4.91	9.79	**2.07**	3.07	0.85
p_hat500-2	500	0.50	36	134.6	14.03	2.12	1.53	3.92	**0.90**	3.26	0.83
p_hat500-3	500	0.75	50	-	5470	1256	349.4	634.4	**55.95**	4.56	0.91
p_hat700-1	700	0.25	11	0.09	2.58	**0.07**	0.14	0.71	0.80	1.88	0.33
p_hat700-2	700	0.50	44	15417	109.8	30.19	12.6	25.80	**4.87**	3.79	0.84
p_hat700-3	700	0.75	62	-	-	-	6187	12178	**1033**	4.64	0.91
p_hat1000-1	1000	0.24	10	1.11	11.93	**0.35**	0.58	2.84	2.53	1.21	0.77
p_hat1000-2	1000	0.49	46	-	7230	1656	412.9	1038	**146.54**	4.21	0.89
p_hat1000-3	1000	0.74	68	-	-	-	-	-	**200760**	5.45	0.93
p_hat1500-1	1500	0.25	12	8.01	206.4	**2.97**	4.31	22.37	15.85	2.19	0.82
p_hat1500-2	1500	0.51	65	-	-	-	61461	105909	**8848**	4.95	0.92
san1000	1000	0.50	15	**0.08**	44.12	3.35	0.74	1.56	1.46	1.67	0.61
san200_0.9_2	200	0.90	60	13.36	1.124	2.92	0.79	1.14	**0.10**	3.39	0.58
san200_0.9_3	200	0.90	44	503.4	78.41	**0.11**	3.43	5.80	0.22	4.86	0.82
san400_0.7_1	400	0.70	40	-	9.99	1.09	0.52	0.67	**0.21**	3.95	0.78
san400_0.7_2	400	0.70	30	3081	28.98	0.21	0.20	**0.09**	**0.09**	1.10	0.19
san400_0.7_3	400	0.70	22	4.47	117.3	2.12	2.04	2.67	**0.75**	2.66	0.83
san400_0.9_1	400	0.90	100	-	729.6	2.5	30.95	56.99	**1.97**	6.58	0.86
sanr200_0.7	200	0.70	18	1.88	1.845	**0.37**	0.39	0.86	0.38	2.40	0.80
sanr200_0.9	200	0.90	42	46593	64.41	204.72	51.57	80.51	**5.72**	5.07	0.91
sanr400_0.5	400	0.50	13	0.97	7.35	**0.52**	0.74	2.13	1.73	2.36	0.57
sanr400_0.7	400	0.70	21	2852	1347	237.26	177.8	429.25	**141.48**	2.52	0.85

une borne supérieure plus petite ou égale à la borne inférieure pour couper les sous-arbres enracinés dans ces noeuds (i.e. $|P|+|C| \leq LB$), que q désigne le nombre de noeuds dans lesquels la technologie MaxSAT réduit avec succès la borne supérieure à la borne inférieure (i.e., $|P|+|C| > LB$, mais $|P|-s+|C| \leq LB$) pour couper les sous-arbres enracinés dans ces noeuds, et que r désigne le nombre de noeuds dans lequel la technologie MaxSAT n'a pas réussi à réduire la borne supérieure à la borne inférieure (i.e., $|P|-s+|C|>LB$). Notons que l'arbre de recherche a $p+q$ feuilles. Dans le tableau 2 et le tableau 3, en plus de temps d'exécution, nous rapportons pour MaxCLQ l'amélioration moyenne s de la borne supérieure dans les ($q+r$) noeuds grâce à la technologie MaxSAT (nous excluons les p feuilles où la technologie MaxSAT n'a pas eu besoin d'etre appliquée), et le taux de réussite $q/(q+r)$ de la technologie MaxSAT pour couper les sous-arbres. Le taux de réussite est très élevé, surtout pour les grands graphes et dense (jusqu'à 93%), expliquant la bonne performance de MaxCLQ.

8 Conclusion

Nous avons proposé un nouveau codage de Maxclique en MaxSAT fondé sur une partition d'un graphe en ensembles indépendants, chaque ensemble indépendant étant codé en une clause souple. Le nombre de clauses souples est nettement plus petit dans le nouveau codage que dans le codage habituel de Maxclique en MaxSAT. Le nouveau codage nous permet d'intégrer naturellement la technologie MaxSAT dans un algorithme de séparation et évalua-

tion pour améliorer la borne supérieure basée sur la partition. Les résultats expérimentaux montrent que de nombreux sous-arbres sont coupés de cette manière et l'algorithme résultant MaxCLQ est très efficace, surtout pour les instances difficiles, et il est capable de fermer une instance DIMACS ouverte.

Nous pensons que l'utilisation de la technologie MaxSAT pour améliorer la borne supérieure de Maxclique est une direction de recherche très prometteuse dans l'avenir. Nous envisageons d'intégrer d'autres technologies MaxSAT effectives dans MaxCLQ pour améliorer encore sa borne supérieure et résoudre les instances Maxclique difficiles telles que celles de l'BHOSLIB benchmark [3].

Références

[1] R. Carraghan, P. M. Pardalos, An exact algorithm for the maximum clique problem. *Operations Research Letters* 9(6) : 375-382 (1990)

[2] T. Fahle, Simple and fast : Improving a branch-and-bound algorithm for maximum clique. In *Proceedings of ESA-2002*, pp. 485-498, 2002.

[3] F. Heras, J. Larrosa, A Max-SAT Inference-Based Preprocessing for Max-Clique, in proceedings of SAT'2008, LNCS 4996, pp. 139-152, 2008.

[4] F. Heras, J. Larrosa, and A. Oliveras, MiniMaxSAT : An efficient weighted Max-SAT solver. *Journal of Artificial Intelligence Research*, 31 :1-32, 2008.

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

[6] C. M. Li and F. Manyà, Max-sat, hard and soft constraints. In A. Biere, H. van Maaren, and T. Walsh, editors, *Handbook of Satisfiability*. Pages 613-631, IOS Press, 2009.

[7] C. M. Li, F. Manyà, and J. Planes, New inference rules for Max-SAT. *Journal of Artificial Intelligence Research*, 30 :321-359, 2007.

[8] C. M. Li, F. Manyà, and Jordi. Planes, Detecting disjoint inconsistent subformulas for computing lower bounds for max-sat, In *Proceedings of AAAI'06*, pages 86-91. AAAI Press, 2006.

[9] C. M. Li, F. Manyà and J. Planes, Exploiting unit propagation to compute lower bounds in branch and bound MaxSAT solvers, In *proceedings of CP'05*, LNCS 3709 Springer, 2005, pp 403-414.

[10] P. R. J. Ostergard, A fast algorithm for the maximum clique problem, *Discrete Applied Mathematics* 120 (2002), 197-207.

[11] P. M. Pardalos, J. Xue, The maximum clique problem. *Journal of Global Optimization* 4 : 301-328, 1994

[12] W. Pullan, H. H. Hoos, Dynamic Local Search for the Maximum Clique Problem. *Journal of Artificial Intelligence Research*, Vol. 25, pp. 159-185, 2006.

3. http ://www.nlsde.buaa.edu.cn/˜kexu/benchmarks/graph-benchmarks.htm

[13] J.-C. Regin, Solving the maximum clique problem with constraint programming. In *Proceedings of CPAIOR'03*, Springer, LNCS 2883, pp. 634-648, 2003.

[14] E. Tomita, T. Kameda, An efficient branch-and-bound algorithm for finding a maximum clique with computational experiments. *J. Glob Optim* (2007) 37 :95-111.

[15] E. Tomita, T. Seki, An efficient branch-and-bound algorithm for finding a maximum clique. In *Proc. Discrete Mathematics and Theoretical Computer Science*. LNCS 2731, pp. 278-289 (2003).