Incremental MaxSAT Reasoning to Reduce Branches in a Branch-and-Bound Algorithm for MaxClique

Chu-Min Li[1,2](✉), Hua Jiang[1], and Ru-Chu Xu[1]

[1] Huazhong University of Sciences and Technology (HUST), Wuhan, China
chu-min.li@u-picardie.fr, jh_hgt@163.com
[2] MIS, Université de Picardie Jules Verne, Amiens, France

Abstract. When searching for a maximum clique of a graph using a branch-and-bound algorithm, it is usually believed that one should minimize the set of branching vertices from which search is necessary. It this paper, we propose an approach called incremental MaxSAT reasoning to reduce the set of branching vertices in three ways, developing three algorithms called DoMC (short for Dynamic ordering MaxClique solver), SoMC and SoMC- (short for Static ordering MaxClique solver), respectively. The three algorithms differ only in the way to reduce the set of branching vertices. To our surprise, although DoMC achieves the smallest set of branching vertices, it is significantly worse than SoMC and SoMC-, because it has to change the vertex ordering for branching when reducing the set of branching vertices. SoMC is the best, because it preserves the static vertex ordering for branching and reduces the set of branching vertices more than SoMC-.

1 Introduction

A clique in an undirected graph $G = (V, E)$, where V is a set of n vertices $\{v_1, v_2, ..., v_n\}$ and E is a set of m edges, is a subset C of V in which every two vertices are adjacent. The maximum clique problem (MaxClique for short) consists in finding a clique of G of the largest size. The size of a maximum clique of G is usually denoted by $\omega(G)$. MaxClique is a very important NP-hard problem, because it is useful in many real-world applications such as bioinformatics and fault diagnosis. A huge amount of effort has been devoted to solve it. In this paper, we focus on exact algorithms for MaxClique based on the Branch-and-Bound (BnB) scheme.

In order to search for a maximum clique in G, a BnB algorithm typically uses a heuristic to order vertices of G to obtain an ordering such as $v_1 < v_2 < v_3 < ... < v_n$, and branches on every vertex v_i for $i = 1, 2, ..., n$ to recursively search for a maximum clique containing v_i in the subgraph G_i induced by $\{v_i, v_{i+1}, ..., v_n\}$. To be efficient, the algorithm maintains a global variable C_{max} to denote the largest clique found so far in G and prunes useless branches in which a clique larger than C_{max} cannot be found. Recent BnB algorithms for MaxClique such

C. Dhaenens et al. (Eds.): LION 9 2015, LNCS 8994, pp. 268–274, 2015.
DOI: 10.1007/978-3-319-19084-6_26

as MCS [5], MaxCliqueDyn [1], and MaxCLQ [3,4] prune useless branches as follows. They first partition the vertices in G into independent sets $D_1, D_2, ..., D_r$ (an independent set is a subset of V in which no two vertices are adjacent). Then if $r > |C_{max}|$, they order the vertices according to their independent set: $v_i < v_j$ if $v_i \in D_p$ and $v_j \in D_q$ and $q < p$. In the vertex ordering $v_1 < v_2 < ... < v_n$ obtained in this way, the vertices in the subset $D_r \cup D_{r-1} \cup ... \cup D_{|C_{max}|+1}$ are the smallest. The algorithms only need to branch on vertices in this subset, since vertices in $D_1, D_2, ...,$ and $D_{|C_{max}|}$ cannot form alone a clique larger than C_{max}.

We call *branching vertices* the vertices that a BnB algorithm needs to branch on. It is a common practice for a state-of-the-art BnB algorithm to reduce as much as possible the number of branching vertices by cleverly ordering vertices. For example, the *Re-NUMBER* procedure in MCS aims at reducing the number of branching vertices by re-organizing the independent sets $D_1, D_2, ...,$ and $D_{|C_{max}|}$ to make them accept more vertices. Consequently, the vertex ordering for branching in the algorithm is dynamic and is different at different search tree nodes.

An exception is the algorithm IncMaxCLQ [2] which uses a static vertex ordering for branching and needs to branch on all vertices of G. Let v_i and v_j be two vertices in G and $v_i < v_j$, the static vertex ordering implies $v_i < v_j$ in any subgraph of G containing v_i and v_j and at every search tree node. The static vertex ordering allows IncMaxCLQ to use an efficient incremental upper bound.

In this paper, we show that deriving the smallest possible set of branching vertices is not necessarily beneficial, that keeping a static vertex ordering probably is more important, and that reducing the number of branching vertices by keeping a static vertex ordering is really beneficial. Concretely, we propose an approach called *incremental MaxSAT reasoning* to reduce the number of branching vertices in three ways, developing three algorithms called DoMC (short for Dynamic ordering MaxClique solver), SoMC and SoMC- (short for Static ordering MaxClique solver), respectively. DoMC uses incremental MaxSAT reasoning to reinforce the Re-NUMBER procedure of MCS, reducing the number of branching vertices more than MCS. Nevertheless, this reduction prohibits any static vertex ordering for branching in DoMC as in MCS. SoMC and SoMC- reduce the number of branching vertices using incremental MaxSAT reasoning by preserving a static vertex ordering. Experimental results show that SoMC, SoMC-, and even IncMaxCLQ that preserves a static vertex ordering but does not reduce the number of branching vertices at all, are significantly better than DoMC, in terms of both search tree size and runtime, although the set of branching vertices in DoMC is smaller. SoMC and SoMC- are also faster than the stat-of-the-art algorithms such as MCS, MaxCliqueDyn, MaxCLQ and IncMaxCLQ. SoMC derives smaller sets of branching vertices than SoMC-, and is better than SoMC-.

2 Incremental MaxSAT Reasoning

Let V' be a subset of V, the subgraph of G induced by V' is defined as $G(V') = (V', E')$, where $E' = \{(v_i, v_j) \in E \mid v_i, v_j \in V'\}$. The set of adjacent vertices of a vertex v in G is denoted by $\Gamma(v) = \{v' | (v, v') \in E\}$. The cardinality $|\Gamma(v)|$ of

$\Gamma(v)$ is called the degree of v. The density of a graph of n vertices and m edges is $2m/(n(n-1))$.

Recent BnB algorithms such as MCS, MaxCliqueDyn, MaxCLQ and IncMax-CLQ partition G into independent sets by sequentially inserting vertices of G into independent sets. Unfortunately, the upper bound given by the independent set partition, called UB_{IndSet} in this paper, may not be tight, because a set of r independent sets may not form a clique of size r. In this case, these independent sets are said *conflicting*. A recent approach proposed in [3,4] uses MaxSAT reasoning to improve UB_{IndSet} by detecting conflicting independent sets, after (implicitly) encoding a MaxClique problem into a partial MaxSAT problem. MaxSAT reasoning as described in [3,4] is not incremental because it is always done from scratch. In this paper, we propose *incremental MaxSAT reasoning* which, given an induced subgraph G' of G with a known upper bound of $\omega(G')$, successively adds vertices of G into G' and detects a conflict in G' after inserting each vertex. The purpose of incremental MaxSAT reasoning is to show that the upper bound of $\omega(G')$ is not increased after inserting these vertices into G'.

Example 1. Consider the graph in Fig. 1 and its subgraph G' induced by $\{v_1, v_2, v_3, v_4\}$. G' is partitioned into 2 independent sets: $\{v_1, v_4\}$, $\{v_2, v_3\}$, so UB_{IndSet}=2 for G'. When inserting v_5 into G', we have a new independent set $\{v_5\}$. Incremental MaxSAT reasoning detects a conflict as follows: assume that each of the three independent sets contributes a vertex to the maximum clique under construction, then v_5 is in the clique, excluding v_1 and v_2 from the clique because they are not adjacent to v_5, so the only remaining v_4 in the first set and the only remaining v_3 in the second set should be in the clique. However, this is not possible, because v_3 and v_4 are not adjacent. So the three independent sets $\{v_1, v_4\}$, $\{v_2, v_3\}$ and $\{v_5\}$ are conflicting.

We add a new vertex $z_1(z_2, z_3)$ into the first (second, third) independent set. Each z_i is unconnected to z_j (for any $j{\neq}i$) and the vertices in the same independent set, but is adjacent to all other vertices in G. If the conflicting independent sets can form a clique of size p without the new vertices, they can form a clique of size $p{+}1$ with the new vertices. So, the new vertices cover exactly one conflict in these independent sets.

Then v_6 is inserted into G'. We have now 4 independent sets: $\{v_1, v_4, z_1\}$, $\{v_2, v_3, z_2\}$, $\{v_5, z_3\}$, and $\{v_6\}$. Incremental MaxSAT reasoning detects a new conflict as follows: the adding of v_6 in the maximum clique under construction excludes v_1 and v_4 from the first set, and v_5 from the third set. However, z_1 and z_3 are not adjacent and cannot both belong to a clique. So, $\{v_1, v_4, z_1\}$, $\{v_5, z_3\}$, and $\{v_6\}$ are conflicting.

The two conflicts detected above for v_5 and v_6 are clearly disjoint because of the adding of z_1, z_2 and z_3, showing that the upper bound of $\omega(G')$ is always 2 after G' includes v_5 and v_6. The second conflict can also be covered by adding a new vertex into each independent set involved in the conflict.

Formally, we define a function IncMaxSAT(G, S, B), where $G = (V, E)$ is a graph with a vertex ordering, S is a subset of vertices that is partitioned into r

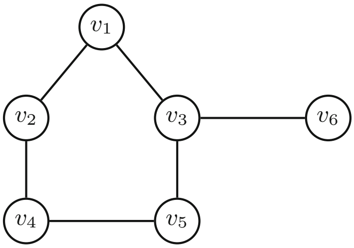

Fig. 1. A simple graph ($\omega(G)$=2) from [3]

Algorithm 1. GetBranches(G, r)

```
Input: G=(V, E), and r: an imposed lower bound for the size of a MaxClique
       of G
Output: a set of branching vertices
begin
    G' ← G; P ← ∅; S ← ∅; B ← ∅;
    while G' is not empty do
        v ← the biggest vertex of G';
        remove v from G';
        if P contains an independent set D in which v is not adjacent to any
        vertex then
            insert v into D; insert v into S;
        else
            if |P|<r then
                create a new independent set D = {v}; P ← P ∪ {D}; insert
                v into S;
            else
                if P contains an independent set D in which v has only one
                adjacent vertex u and u can be inserted into another
                independent set D' then
                    move u from D to D'; insert v into D; insert v into S;
                else B ← {v} ∪ B;
    B ← IncMaxSAT(G, S, B);
    return the set of all vertices of G smaller than or equal to the biggest
    vertex in B.
```

---

independent sets, and $B = V \setminus S$ is a set of branching vertices to be reduced. The function successively inserts vertices of B (from the biggest vertex to the smallest one in the predefined vertex ordering) into S, and detects a disjoint conflict in S for each inserted vertex. The detected conflicts show that S with the inserted vertices cannot form a clique of size larger than r. The function stops as soon as it fails to detect a conflict when inserting a vertex v into S, and returns the set of remaining vertices in B (including v).

Algorithm 2. SoMC(G, C, C_{max}), a BnB algorithm for MaxClique

Input: G=(V, E), clique C under construction, and the largest clique C_{max} found so far

Output: $C \cup C'$, where C' is a maxclique of G, if $|C \cup C'| > |C_{max}|$; C_{max} otherwise

begin
  if $|V|=0$ **then** return C;
  $B \leftarrow$ GetBranches(G, $|C_{max}|$-$|C|$);
  if $B=\emptyset$ **then** return C_{max};
  $S \leftarrow V \backslash B$;
  for $i:=|B|$ downto 1 **do**
    $C_1 \leftarrow$ SoMC($G(\Gamma(b_i) \cap S)$, $C \cup \{b_i\}$, C_{max});
    $S \leftarrow \{b_i\} \cup S$;
    if $|C_1| > |C_{max}|$ **then** $C_{max} \leftarrow C_1$;
  return C_{max};

Table 1. Median runtimes in seconds and tree sizes in thousands for random graphs (computed by solving 51 graphs at each point). The points where fewer than 26 graphs are solved within 5000 s are marked by "-". "Dyn" stands for MaxCliqueDyn.

N	D	Dyn	MCS	MaxCLQ	IncMaxCLQ		SoMC		DoMC		SoMC-	
		Time	Time	Time	Time	Tree size	Time	Tree size	Time	Tree size	Time	Tree size
200	0.80	4.56	2.26	1.63	1.27	111	**0.92**	**97.1**	1.33	113	1.19	121
200	0.90	61.87	34.18	9.18	6.22	305	**4.64**	**263**	6.86	341	5.98	299
200	0.95	28.45	13.41	1.59	0.74	22.2	**0.54**	**20.7**	0.67	22.4	0.71	22.1
300	0.70	7.91	6.38	6.12	5.62	642	**3.68**	**503**	5.62	567	4.77	646
300	0.80	269.5	203.1	117.3	105.7	8166	**74.12**	**6611**	143.5	10326	99.18	8372
300	0.90	-	-	-	-	-	**4169**	**182516**	-	-	-	-
400	0.60	4.70	4.19	5.94	4.99	685	**3.53**	**521**	4.84	556	4.58	697
400	0.70	99.76	96.79	89.96	86.43	8590	**58.15**	**6687**	90.56	8416	75.24	8488
400	0.80	-	-	4877	4986	475219	**2834**	**269528**	-	-	3868	345920
500	0.50	1.85	**1.62**	2.96	2.42	519	1.66	410	1.94	**294**	1.93	470
500	0.60	26.15	23.27	29.93	26.97	3822	**17.31**	**2688**	25.66	2875	21.08	3478
500	0.70	915.8	916.1	766.2	883.8	81697	**646.2**	**61687**	905.5	78280	709.4	79602
1000	0.30	0.75	**0.70**	2.51	1.27	374	1.02	217	1.04	**163**	1.23	354
1000	0.40	8.47	7.57	19.15	8.90	2279	**6.59**	1934	8.68	**1556**	7.20	2083
1000	0.50	176.9	167.2	303.2	214.3	40154	**139.9**	27445	188.8	**25256**	164.6	34645

3 Applying Incremental MaxSAT Reasoning to Reduce the Number of Branching Vertices

A BnB algorithm always searches for a maximum clique of size larger than a given lower bound r in $G = (V, E)$. Assuming V is totally ordered, we define the function GetBranches(G, r) in Algorithm 1 that returns a set of branching vertices B by showing vertices in $V \setminus B$ cannot form a clique of size larger than r. The function works in two phases: in the first phase, r independent sets are

Table 2. Runtimes in seconds and tree sizes in thousands for DIMACS instances that are solved by at least one solver in 10^5 s, excluding the instances solved by all solvers in 10 s. "-" stands for instances that cannot be solved in 10^5 s. "Dyn" stands for MaxCliqueDyn.

Instance	N	D	Dyn	MCS	MaxCLQ	IncMaxCLQ		SoMC		DoMC		SoMC-	
			Time	Time	Time	Time	Tree size	Time	Tree size	Time	Tree size	Time	Tree size
brock400_1	400	0.74	466.0	379.7	339.2	222.3	18906	**147.8**	**14541**	345.2	52167	189.5	18826
brock400_2	400	0.74	192.1	166.2	**105.9**	170.2	14474	109.9	**11160**	303.7	22489	146.7	14367
brock400_3	400	0.74	371.6	256.2	102.4	204.6	17735	**80.17**	**8022**	267.7	8275	108.5	10272
brock400_4	400	0.74	185.7	138.2	125.3	159.2	13319	**105.7**	**10307**	196.5	10746	140.3	13337
brock800_1	800	0.65	5988	5209	4889	8830	890495	**2142**	**233383**	9063	810594	2838	298022
brock800_2	800	0.65	5349	4686	4857	11210	1125211	**2139**	**221625**	8315	538086	2872	287018
brock800_3	800	0.65	3455	3208	3452	4221	398861	**895.8**	**105807**	7628	392722	1161	131127
brock800_4	800	0.65	2691	2259	3441	5832	547564	**1483**	**173233**	4589	251810	1947	217924
C2000.5	2000	0.50	-	-	-	61009	9901896	**41222**	6606311	46381	**5704024**	48332	8604571
C250.9	250	0.89	2376	2074	298.2	278.9	12066	**202.1**	**10055**	372.6	16361	264.5	11707
DSJC1000.5	1000	0.50	185.2	169.6	295.8	226.3	38431	**134.9**	26826	196.2	**26604**	154.9	33278
gen400_p0.9_55	400	0.90	-	37220	-	1.23	**4.01**	**1.17**	4.03	1.56	5.13	**1.17**	4.03
gen400_p0.9_65	400	0.90	-	96567	26134	0.34	3.09	**0.29**	**3.09**	0.37	3.24	0.30	3.09
gen400_p0.9_75	400	0.90	-	-	1372	0.27	6.64	**0.17**	**3.34**	0.18	3.52	**0.17**	**3.34**
hamming10-2	1024	0.99	49.73	0.19	**0.06**	32.65	131	34.49	131	34.59	131	34.43	131
keller5	776	0.75	-	-	5376	**141.6**	**2092**	192.4	7818	344.3	10837	199.3	8106
MANN_a45	1035	0.99	1712	63.09	20.04	115.8	218	15.49	85.4	**13.23**	**75.7**	15.48	86.2
p_hat1000-2	1000	0.49	276.6	131.6	219.9	48.75	1855	**33.87**	**1391**	53.38	1778	44.69	1612
p_hat1000-3	1000	0.75	-	-	-	42244	1028854	**27727**	**618456**	-	-	36521	804780
p_hat1500-2	1500	0.51	-	10448	15138	2165	45143	**1322**	**26354**	4023	77650	1829	35299
p_hat500-3	500	0.75	235.1	79.23	81.04	22.02	941	**15.59**	**704**	23.59	823	19.20	792
p_hat700-3	700	0.75	3946	1586	1009	269.5	7544	**178.9**	**4954**	434.8	10891	233.3	6121
sanr200_0.9	200	0.90	32.56	19.68	6.08	2.71	128	**1.97**	**107**	3.63	170	2.39	119
sanr400_0.7	400	0.70	110.0	99.69	98.56	104.7	10607	**70.06**	**8390**	100.1	15146	90.26	10658

formed using the coloring process of MCS with the Re-NUMBER procedure, and an initial set B of branching vertices is obtained; in the second phase, incremental MaxSAT reasoning is applied to eliminate the biggest vertices of B from which any clique of size larger than r cannot be found.

The BnB algorithm SoMC depicted in Algorithm 2 calls the GetBranches function to obtain a reduced set B of branching vertices $\{b_1, b_2, ..., b_{|B|}\}$ and successively branches on vertex b_i (for $i = |B|, |B|-1, ..., 1$) to search for a maximum clique containing b_i in the subgraph of G induced by $\{b_i, b_{i+1}, ..., b_{|B|}\} \cup S$, where $S = V \setminus B$, and B is ordered as V. Note that for any $b \in B$ and any $v \in S$, we have $b < v$, meaning that the set $\{b_i, b_{i+1}, ..., b_{|B|}\} \cup S$ can never contain a vertex smaller than b_i. This fact is exploited in the implementation of SoMC to speed up search using an incremental upper bound as in IncMaxCLQ. See [2] for details.

The GetBranches function returns the set of all vertices of G smaller than or equal to the biggest vertex in B, which is larger than the set given by IncMaxSAT(G, S, B) in line 15, because some vertices smaller than the biggest vertex in B could be inserted into S by the independent set partition, but are included in the set returned by the GetBranches function in line 16. We can modify GetBranches to make it simply return IncMaxSAT(G, S, B) in line 15, giving another BnB algorithm called DoMC, in which the set of branching vertices is smaller, but the static vertex ordering for branching is not preserved any more, because the set $\{b_i, b_{i+1}, ..., b_{|B|}\} \cup S$ can now contain some vertices smaller than b_i when DoMC branches on b_i.

Another possibility to make S not contain vertices smaller than any vertex in B is to stop the independent set partition in the GetBranches function as soon as a vertex cannot be inserted into the r independent sets (i.e. line 14 is replaced by "**else** *break*", then the function returns IncMaxSAT$(G, S, V \setminus S)$ in line 15). This modification gives the BnB algorithm SoMC- which also exploits the incremental upper bound as SoMC and IncMaxCLQ, thanks to the preserved static vertex ordering for branching.

Note that SoMC, SoMC- and DoMC are the same except the modifications described above. They share the same implementation and use the same vertex ordering as in IncMaxCLQ to partition G into independent sets in the GetBranches function. We now compare them, as well as MaxCliqueDyn, MCS, MaxCLQ, and IncMaxCLQ on standard MaxClique benchmarks on an Intel Xeon CPU X5460@3.16 GHz under Linux with 16 GB of memory. All solvers were compiled using gcc/g++ -O3.

Tables 1 and 2 show the runtimes (in seconds) of all algorithms and search tree sizes (in thousands) of IncMaxCLQ, SoMC, SoMC- and DoMC. Except few graphs, the search trees of DoMC are larger than IncMaxCLQ, SoMC and SoMC- that keep static vertex ordering for branching, although DoMC derives the smallest set of branching vertices. SoMC- is better than IncMaxCLQ because IncMaxCLQ does not reduce the number of branching vertices at all, while SoMC- does. SoMC is better than SoMC- because SoMC derives smaller sets of branching vertices than SoMC-.

SoMC and SoMC- are faster than MaxCLQ, MaxCliqueDyn, and MCS.

References

1. Konc, J., Janezic, D.: An improved branch and bound algorithm for the maximum clique problem. Commun. Math. Comput. Chem. **58**, 569–590 (2007)
2. Li, C.M., Fang, Z.W., Xu, K.: Combining MaxSAT reasoning and incremental upper bound for the maximum clique problem. In: Proceedings of the 2013 IEEE 25th International Conference on Tools with Artificial Intelligence (ICTAI2013), pp. 939–946 (2013)
3. Li, C.M., Quan, Z.: An efficient branch-and-bound algorithm based on maxsat for the maximum clique problem. In: Proceedings of the 24th AAAI, pp. 128–133 (2010)
4. Li, C.M., Quan, Z.: Combining graph structure exploitation and propositional reasoning for the maximum clique problem. In: Proceedings of the 22th ICTAI, pp. 344–351 (2010)
5. Tomita, E., Sutani, Y., Higashi, T., Takahashi, S., Wakatsuki, M.: A Simple and Faster Branch-and-Bound Algorithm for Finding a Maximum Clique. In: Rahman, M.S., Fujita, S. (eds.) WALCOM 2010. LNCS, vol. 5942, pp. 191–203. Springer, Heidelberg (2010)