Conflict Directed Clause Learning for the Maximum Weighted Clique Problem

Emmanuel Hebrard[1] and **George Katsirelos**[2]
[1] LAAS-CNRS, Université de Toulouse, CNRS, Toulouse, France
[2] MIAT, INRA, Toulouse, France
hebrard@laas.fr, gkatsi@gmail.com

Abstract

The maximum clique and minimum vertex cover problems are among Karp's 21 NP-complete problems, and have numerous applications: in combinatorial auctions, for computing phylogenetic trees, to predict the structure of proteins, to analyse social networks, and so forth.

Currently, the best complete methods are branch & bound algorithms and rely largely on graph colouring to compute a bound.

We introduce a new approach based on SAT and on the "Conflict-Driven Clause Learning" (CDCL) algorithm. We propose an efficient implementation of Babel's bound and pruning rule, as well as a novel dominance rule. Moreover, we show how to compute concise explanations for this inference. Our experimental results show that this approach is competitive and often outperforms the state of the art for finding cliques of maximum weight.

1 Introduction

A clique of a graph is a subset of vertices whose induced subgraph is complete. Finding a clique of maximum weight has numerous applications. For instance computing a maximum set of characters defining a perfect phylogeny is akin to solving a maximum clique problem [Day and Sankoff, 1986]. Similarly, predicting the structure of a protein [Samudrala and Moult, 1998], comparing the structures of two proteins [Strickland *et al.*, 2005], or computing maximum error-detection codes [MacWilliams and Sloane, 1978] can all be seen as finding a maximum weighted clique in a graph.

As a result, a wide range of algorithms and heuristics have been proposed for tackling the maximum clique problem [Bomze *et al.*, 1999]. There are relatively fewer algorithms for the weighted version, and even fewer complete algorithms. State of the art complete approaches are often based on branch & bound, sometimes used within a Russian doll scheme, as in `cliquer` [Östergård, 2001; 2002]. Dual bounds are often obtained by colouring the graph: if there is a k-colouring then there is no k-clique.

In [Fang *et al.*, 2016; Jiang *et al.*, 2017; 2018], a simple generalisation of the graph colouring bound to the weighted case is improved using techniques from MaxSAT. In another generalisation proposed in [Babel, 1994], a vertex is coloured using as many distinct colours as its weight. The cardinality of such a *multicolouring* is an upper bound on the weight of a clique. However the complexity of Babel's algorithm depends on the number of colours, hence it can be impractical when the weights are large. Tavares proposed an algorithm whose complexity does not depend on the weights [Tavares, 2016]. It iteratively computes an independent set of the graph and colours its vertices with the same set of k colours where k is the smallest of these nodes' weights.

We propose a novel approach based on constraint programming and on the "Conflict-Driven Clause Learning" (CDCL) algorithm. We use a method similar to Tavares' to compute a dual bound except that we compute several independent sets at once. On a graph with n vertices and m edges, our method runs in $O(n^3)$ time, irrespective of the weight function. Next, we introduce a pruning rule and a dominance rule both based on this bound and both running in $O(n^2)$. We then show how to compute short clauses to explain this inference in $O(mn)$.

Experimental results on several standard benchmarks show that this approach is very promising and compares favourably with state-of-the-art methods on many families of instances.

2 Problem Description

Let $G = (V, E)$ be a graph with V a set of vertices and E a set of edges, and let w be a function mapping every $v \in V$ to an integer $w(v)$. For a set $W \subseteq V$, $w(W) = \sum_{v \in W} w(v)$. We write $N(v) = \{u \mid (u, v) \in E\}$ for the *neighbourhood* of v in G, $N^+(v) = N(v) \cup \{v\}$, and $N(S) = \cup_{v \in S} N(S)$.

A clique of G is a subset of V such that every pair of vertices share an edge. The *Maximum Weighted Clique* problem asks for the clique C maximising $w(C)$. An independent set is a subset $\mathcal{I}$ of V such that no pair of vertices in $\mathcal{I}$ are adjacent. A vertex cover is a subset $\mathcal{V}$ of V that contains at least one vertex adjacent to every edge in E. If $\mathcal{I}$ is a clique of G, it is also an independent set of the complement graph $\overline{G} = (V, \{(u, v) \mid u \neq v \wedge (u, v) \notin E\})$ and $\mathcal{V} = V \setminus \mathcal{I}$ is a vertex cover of $\overline{G}$. These equivalences preserve optimality. Here, we use the viewpoint of independent set / vertex cover.

3 Overview of the Algorithm

We propose a constraint programming model for the maximum independent set problem. We have one variable x_v per vertex v in the graph. The positive literal x_v, stands for "$v \in \mathcal{V}$" and its negation $\overline{x_v}$ stands for "$v \in \mathcal{I}$". The model contains a single constraint WEIGHTEDIS.

$$\text{WEIGHTEDIS} \quad (x_v, G, w, k) \tag{1}$$

The constraint is satisfied by an assignment if $\{v \mid x_v \text{ is false}\}$ is an independent set of G of weight at least k and is clearly NP-hard. The propagator for this constraint aggregates the methods that we develop in the next sections. Throughout, we make use of the fact that it maintains the current vertex cover $\mathcal{V}$ and independent set $\mathcal{I}$, as well as the residual graph, which is the subgraph of G induced by $V \setminus (\mathcal{V} \cup \mathcal{I})$. When a variable x_v is made true, v is added to the current vertex set $\mathcal{V}$, and conversely, when it is made false, it is added to $\mathcal{I}$ and $N(v)$ added to $\mathcal{V}$. In both cases, the residual graph is updated accordingly.

The constraint WEIGHTEDIS handles all aspects of the problem: computing a dual bound (section 4), as well as pruning and propagating dominance rules (section 5). It additionally embeds a primal heuristic (section 7) which may compute solutions before the solver reaches a leaf node in its backtracking search. Notice that the reasoning components described below (computation of the dual bound, dominance and pruning) are not idempotent and are called until a fix point is reached in each node.

Our methods are used in the context of the *Conflict-driven clause learning* (CDCL) algorithm [Marques Silva and Sakallah, 1999; Moskewicz *et al.*, 2001], which is an algorithm for the Boolean Satisfiability problem. We assume familiarity with both for reasons of space and refer the interested reader to Mitchel's primer [Mitchell, 2005] for an in-depth description. The CDCL algorithm usually handles propositional formulas in conjunctive normal form (CNF), i.e., a conjunction of clauses. However, it can be used with formulas which are not in CNF, in particular with arbitrary CSPs [Katsirelos and Bacchus, 2005]. To do this, all constraints have to annotate prunings with clausal reasons, or *explanations*. Moreover, when a constraint becomes unsatisfiable it must again generate an explanation for this. In this setting, the constraints of the problem (in our case WEIGHTEDIS) perform the same task as unit propagation on a potentially exponentially large set of clauses, but do so in polynomial time, and only generate the actual clauses as needed to perform conflict analysis. Hence, in section 6, we describe how we can generate clausal explanations for the inference performed by the various components. After the first conflict, the solver will maintain a set of learned clauses and propagation of WEIGHTEDIS will be interleaved with unit propagation on the learned clauses.

We summarize the propagator for WEIGHTEDIS in algorithm 1. It gets the original graph G, the weight function w, the lower (primal) bound k and the current partial assignment A as arguments and returns a clause signifying a conflict or no conflict but potentially pruning some variables. It starts by ensuring that no two neighboring vertices can be in $\mathcal{I}$ and then proceeds to use the other components. For simplicity, we show it as recomputing $\mathcal{V}$, $\mathcal{I}$ and the residual graph each time it is invoked, but this is in fact maintained incrementally during search. DUALBOUND is algorithm 2, MCCPROPAGATE is algorithm 3, while EXPLAINBOUND is algorithm 4. The procedures $\text{PRUNE}_{\mathcal{V}}$ and $\text{PRUNE}_{\mathcal{I}}$ are much simpler and described in the text in section 6. Finally, we describe a dedicated branching heuristic in section 8.

Algorithm 1: WEIGHTEDIS$(G = (V, E), w, k, A)$

```
I = {v | x̄_v ∈ A} ;
V = {v | x_v ∈ A} ;
if N(I) \ V ≠ ∅ then
    forall v ∈ N(I) \ V do
        u = Pick a vertex from N(v) ∩ I ;
        set x_v to true using reason (x_v ∨ x_u);
        V ← V ∪ {v}
G_r = G |_{V\(I∪V)} ;
fixpoint = false ;
while ¬ fixpoint do
    fixpoint ← true ;
    M = DUALBOUND(G_r) ;
    if w(I) + |M| ≤ k then
        return EXPLAINBOUND(G_r, w, A, M, k − w(I))
    else
        (D, P) = MCCPROPAGATE(G_R, M) ;
        forall v ∈ D do PRUNE_I(x̄_v, G_R, w, I, V, M) ;
        forall v ∈ P do PRUNE_V(x_v, G_R, w, I, V, M) ;
        if D ≠ ∅ ∨ P ≠ ∅ then fixpoint ← false ;
return no-conflict ;
```

4 Dual Bound

A standard dual bound for the maximum independent set of a graph $G = (V, E)$ relies on a *clique cover*, i.e., a partition of V into a set of cliques $\mathcal{C}$ such that $\bigcup_{C \in \mathcal{C}} C = V$. Indeed, at most one vertex of each clique can be in the independent set.

This bound can be transposed to the weighted case by considering a *clique multicover*, that is, a collection of cliques $\mathcal{M}$ such that every vertex v belongs to $w(v)$ cliques [Babel, 1994][1]. Let $\mathcal{I}$ be an independent set of G. No two vertices in $\mathcal{I}$ share a clique, therefore any *clique multicover* has cardinality at least $\sum_{v \in \mathcal{I}} w(v)$. It follows that the cardinality $|\mathcal{M}|$ of a clique multicover $\mathcal{M}$ is an upper bound of the weight of any independent set of G. A clique multicover is a multiset, and we shall use the notation $m(C)$ for the number of copies of element C in the multiset (which shall be evident from the context). Given two multisets $\mathcal{M}$ and $\mathcal{M}'$, $\mathcal{M} \uplus \mathcal{M}'$ is their union. $\{C^w\}$ is the multiset containing w copies of C.

We propose in algorithm 2 a slight variation over the algorithm introduced in [Tavares, 2016] to compute a clique multicover efficiently, even when the weights are arbitrarily large. The main difference is that instead of considering one clique at a time, we find a complete clique cover $\mathcal{C}$ (i.e., a colouring of the complement graph). Doing so reduces the number of iterations and allows us to use a known algorithm

[1] Viewed from the clique/multicolouring angle in that paper.

Algorithm 2: DUALBOUND$(G = (V, E))$

$\mathcal{M} \leftarrow \emptyset$;
while $V \neq \emptyset$ **do**
1 $\quad \mathcal{C} \leftarrow CliqueCover(G, V)$;
$\quad$ **if** $|\mathcal{C}| = |V|$ **then return** $\mathcal{M} \uplus \{C^{w(C)} \mid C \in \mathcal{C}\}$;
2 $\quad$ **foreach** $C \in \mathcal{C} \mid 1 < |C|$ **do**
$\quad\quad \alpha \leftarrow \min_{v \in C} w(v)$;
3 $\quad\quad \mathcal{M} \leftarrow \mathcal{M} \uplus \{C^{\alpha}\}$;
4 $\quad\quad$ **foreach** $v \in C$ **do** $w(v) \leftarrow w(v) - \alpha$;
$\quad V \leftarrow V \setminus \{v \in V \mid w(v) = 0\}$;
return $\mathcal{M}$;

to compute the clique cover. DSatur [Brélaz, 1979] is too slow to compute at every node, so we use instead a simpler algorithm. Like DSatur it greedily adds a vertex to the first possible clique, but considers the vertices in the inverse of the degeneracy order [Lick and White, 1970], computed statically. This runs in time $O(|V||\mathcal{C}|)$ plus $O(|V|)$ bitset AND operations to maintain the set of potential vertices in each clique. For each non-singleton clique C of the clique cover $\mathcal{C}$ computed in line 1, we add $\min_{v \in C} w(v)$ copies of C to $\mathcal{M}$ (line 3) and subtract $\min_{v \in C} w(v)$ from the weight of each vertex in C (line 4). Then we continue finding new clique covers of the graph induced by the vertices which have non-zero weight, and adding the corresponding number of cliques to $\mathcal{M}$, until all cliques in the cover have size 1. The correctness of the bound is immediate from the fact that it computes a sound clique multicover.

With respect to complexity, observe first that we do not actually create copies of each clique, but rather maintain a count of copies of each distinct clique. Therefore, the number of cliques that we store, as well as the complexity of algorithm 2, is independent of the magnitude of the weights. Next, observe that, ignoring the calls to CLIQUECOVER in Line 1, it runs in $O(|V|^2)$ time since the inner loop in line 2 is in $O(|V|)$. The dominant factor for the computational complexity is therefore Line 1. Consider first a variant of the algorithm where we do not postpone the insertion of singleton cliques, i.e., we remove the condition $1 < |C|$ from the loop in line 2. Then, each clique consumes the remaining weight of at least one clique, so $O(|V|)$ cliques are inserted in $\mathcal{M}$ over all iterations. Therefore, the cumulative complexity ignoring the bitet operations is $O(|V||\mathcal{C}_1| + \ldots + |V||\mathcal{C}_k|) = O(|V|^2)$, hence dominated by the $O(|V|^2)$ bitset AND operations.

The reason for postponing the insertion of singleton cliques into the cover is that the constraint implied by a singleton clique is the tautology that we can use at most one vertex from that singleton clique in $\mathcal{I}$. If we postpone inserting it into the cover, this vertex may appear in a larger clique in a later iteration. But doing so destroys the property that the number of cliques is $O(|V|)$ and there can in fact be as many as $O(|V|^2)$ (for example, in a star graph where the center of the star has weight $|V|$ and the rest of the vertices have weight 1), giving a total complexity of $O(|V|^3)$. In practice, however, we have observed no significant slowdown from postponing inserting singleton cliques, while the dual bound is consistently improved. For example, in the star graph described above we compute a bound of $2|V| - 1$ if we do not postpone insertion of singleton cliques, but if we do we get $|V|$, which is tight.

Example 1. *Consider the graph illustrated in Figure 1a. algorithm* DUALBOUND *starts with the 2-clique cover represented using solid line for the first clique and dashed lines for the second. It then creates 3 copies of each of $\{a, b, c\}$ and $\{d, e, f\}$ and considers the residual graph shown in Figure 1b. Next, 6 copies of clique $\{a, d, f\}$ are created, yielding the residual graph shown in Figure 1c. Then, 1 copy of clique $\{a, f\}$ is created, yielding isolated vertices hence 3 copies of $\{f\}$ and 2 copies of $\{b\}$, for a total cardinality of 18.*

5 Pruning and Dominance

We perform pruning based on Babel's bound [Babel, 1994]. Consider a clique multicover $\mathcal{M}$ and a vertex v. If we place v in the independent set $\mathcal{I}$, we must place $N(v)$ in the vertex cover $\mathcal{V}$. The residual graph G' will then be over the vertices $V \setminus N^+(v)$. The set $\mathcal{M}' = \{C \setminus N^+(v) \mid C \in \mathcal{M} \wedge C \setminus N^+(v) \neq \emptyset\}$ is a clique multicover of G'. Therefore, if $w(v) + |\mathcal{M}'|$ is smaller than the current primal bound, it means that v must be placed in $\mathcal{V}$. The difference between the original dual bound and the bound subject to v being in the independent set is $|\mathcal{M}| - w(v) - |\mathcal{M}'|$.

By construction, $\mathcal{M}'$ cannot contain any clique including v, as they become empty in G', hence $|\mathcal{M}'| \leq |\mathcal{M}| - w(v)$. The rest of the difference comes from cliques in $\mathcal{M}$ that do not contain v but are not in $\mathcal{M}'$ because they are contained in the neighbourhood of v. Hence, we can compute the marginal cost of adding v to the independent set as the number of cliques contained in $N(v)$.

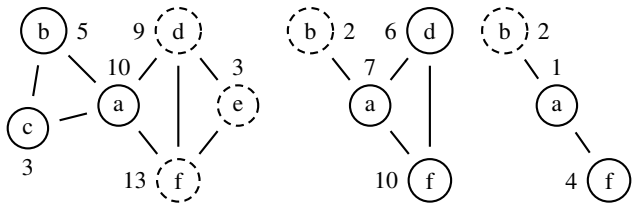
Proposition 1 (Babel's rule)**.** *If the following relation holds:*

$$ub - \sum_{C \in \mathcal{M} | C \subseteq N(v)} m(C) \leq lb \tag{2}$$

then no maximum weight independent set of G contains v.

Example 2. *Consider in Figure 1 vertex d and the clique multicover computed by algorithm 2 (figures 1(a) - 1(c)). Its neighbourhood is $\{a, f, e\}$, which contains 1 copy of $\{a, f\}$ and 3 copies of $\{f\}$, for a total of 4 cliques. Subtracted from the upper bound of 18, this means that no independent set with d can have weight larger than 14.*

This pruning rule is stronger than the rule proposed for use in preprocessing in [Jiang *et al.*, 2017], which is the weighted version of the Buss rule. In terms of the vertex cover, it says that if $lb \geq w(V) - w(N(v))$, then we can put v in the vertex cover $\mathcal{V}$. This is subsumed by pruning based on Babel's rule.



(a) clique cover 1 (b) clique cover 2 (c) clique cover 3

Figure 1: Computing clique multicover.

Proposition 2. *The Buss rule is subsumed by Babel's rule*

Proof. The Buss rule implies $lb \geq w(v) + w(V \setminus N^+(v))$. However, since $\mathcal{M}'$ is a clique multicover of $V \setminus N^+(v)$ we have $|\mathcal{M}'| \leq w(V \setminus N^+(v))$. By substituting the latter inequality into the former, $lb \geq w(v) + |\mathcal{M}'|$ (the definition of Babel's rule). □

Dual reasoning yields the following novel dominance rule:

Proposition 3. *If the following relation holds:*

$$\sum_{C \in \mathcal{M} | C \cap N(v) \neq \emptyset} m(C) \leq w(v) \tag{3}$$

then v belongs to a maximum weight independent set of G.

Proof. Consider a maximum weight independent set $\mathcal{I}$ that does not contain v and let $\mathcal{I}'$ be the independent set obtained by adding v to, and removing all its neighbours from, $\mathcal{I}$:

$$\mathcal{I}' = \mathcal{I} \setminus N(v) \cup \{v\}$$

We have $w(\mathcal{I}) - w(\mathcal{I}') = w(\mathcal{I} \cap N(v)) - w(v)$. We prove $w(\mathcal{I}') \geq w(\mathcal{I})$ which means $w(\mathcal{I} \cap N(v)) \leq w(v)$. By way of contradiction, suppose $w(\mathcal{I} \cap N(v)) > w(v)$. By Babel's reasoning, since $\mathcal{I} \cap N(v)$ is an independent set, any clique multicover of $N(v)$ has cardinality at least $w(\mathcal{I} \cap N(v))$ and therefore strictly higher than $w(v)$. However, if equation 3 holds, then there exists a multicover of the neighbourhood of v whose cardinality is less than or equal to $w(v)$, a contradition to our assumption. Therefore, we have $w(\mathcal{I}') \geq w(\mathcal{I})$ and $\mathcal{I}$' is also a maximum weight independent set. □

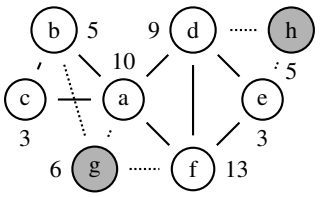
Example 3. *Consider again the graph of Figure 1a. The clique multicover shown in figures 1a, 1b and 1c contains 18 cliques. However, there are 5 cliques (2 copies of $\{b\}$ and 3 copies of $\{f\}$) that do not overlap with $N(f)$. Therefore there are 13 cliques in $\mathcal{M}$ overlapping with $N(f)$.*

Since $w(f) = 13$, we can move f into the independent set, hence all other vertices but b and c into the vertex cover. Another iteration of the dominance rule gives the optimal solution in this case: $\mathcal{V} = \{a, c, d, e\}$ and $\mathcal{I} = \{b, f\}$.

algorithm 3 returns, for a graph $G = (V, E)$ and clique multicover $\mathcal{M}$, the set of vertices dominated and the set of vertices pruned with respect to $\mathcal{M}$. It checks if equations 2 and 3 hold for all vertices in $O(|V|^2)$. Indeed, the weight of at least one vertex becomes null in each new clique added to $\mathcal{M}$, hence the number of distinct cliques is in $O(|V|)$. Moreover, checking if v has a neighbour in C (line 1) and if C is contained in $N(v)$ (line 2) are in $O(1)$ because we can reuse the neighbourhood (resp. potential vertices) of each clique computed in algorithm 2.

Algorithm 3: MCCPROPAGATE($G = (V, E), \mathcal{M}, lb, ub$)

```
pruned ← ∅;
dominated ← ∅;
foreach v ∈ V do
    ub_v ← ub;
    loss_v ← 0;
    foreach distinct clique C ∈ M do
1       if v ∈ N(C) then loss_v ← loss_v + m(C) ;
2       else if C ⊆ N(v) then ub_v ← ub_v − m(C) ;
    if ub_v ≤ lb then pruned ← pruned ∪ {v};
    if loss_v ≤ w(v) then dominated ← dominated ∪ {v};
return dominated, pruned;
```



(a) residual graph

$\{a, b, c\}$ (3)
$\{d, e, f\}$ (3)
$\{a, d, f\}$ (6)
$\{a, f, \mathbf{g}\}$ (1)
$\{b, \mathbf{g}\}$ (2)
$\{f, \mathbf{g}\}$ (3)

(b) clique multicover

Figure 2: Explanation minimisation

6 Explanation

We show here how to concisely explain a failure triggered by the dual bound. Let k be the weight of the maximum independent set of G =(V,E) found so far and $\mathcal{I}, \mathcal{V}$ be the current independent set and vertex cover, and $\mathcal{M}$ a clique multicover of the residual graph. A failure is triggered if

$$ub = w(\mathcal{I}) + |\mathcal{M}| \leq k \tag{4}$$

The method we propose starts from the trivial explanation clause that excludes *any* one decision or deduction, that is: $\bigvee_{v \in \mathcal{V}} \overline{x_v} \vee \bigvee_{v \in \mathcal{I}} x_v$, and minimises it by removing literals that are not necessary. Note that the explanation need only justify that a subset of literals in the current partial assignment is enough to yield a residual graph verifying inequality 4. Since the residual graph has no edges that are not present in the original graph, the clauses do not have to justify the existence of all the cliques in the clique multicover. The key idea that we exploit is that if we can move a vertex v from $\mathcal{V}$ to $\mathcal{M}$ in such a way that inequality 4 still holds, then we can remove the literal $\overline{x_v}$ from the explanation clause.

Example 4. *Consider again the recurring example, and suppose that this is the residual graph obtained after two decisions: adding the vertices h and g to the vertex cover $\mathcal{V}$ as shown in Figure 2. Moreover, suppose that we already discovered an independent set of weight 20, the primal bound. Since the dual bound is $0 + |\mathcal{M}| = 18$ this is a failure and the trivial explanation clause is $x_h \vee x_g$.*

However, $w(g) = 6$ can be distributed over $\{b\}, \{f\}$ and $\{a, f\}$, as shown in Figure 2b, to obtain a smaller vertex cover with same primal-dual gap. Therefore, we can deduce the unit literal x_h. Observe that if the primal bound had been 23, we could have created 5 copies of the singleton clique $\{h\}$, hence proving that 23 is the optimal solution by deriveing the empty clause.

Moreover, we make the simple observation that for any vertex $v \in \mathcal{I}$, we can either remove the literal x_v from the clause, or exclude every literal $\overline{x_u}$ such that $u \in N(v)$. Indeed, $\overline{x_v}$ and the the constraint WEIGHTEDIS directly imply x_u for any $u \in N(v)$. In the other direction, if $\bigwedge_{u \in N(v)} x_u$ is true,

Algorithm 4: EXPLAINBOUND$(G, w, A, \mathcal{M}, L)$

```
A ← stack of all decisions and deductions;
R ← ∅;
Γ ← ∅;
while A ≠ ∅ do
    pop p from A;
1   if p is a neighborhood literal then Γ ← Γ ∪ {p};
2   else if p is positive then
        c ←MARGINAL(v(p), M, G, w);
        if c + |M| > L then R ← R ∪ {p̄} ;
        else INSERT(v(l), M, G, w) ;
3   else
        if Σ_{u∈Γ} MARGINAL(u, M, G, w) + |M| ≤ L then
            while Γ ≠ ∅ do
                pick u of mininum
                  c =MARGINAL(u, M, G, w) and remove it
                  from Γ;
                if c + |M| ≤ L then R ← R ∪ p̄ ;
                else INSERT(u, M, G, w) ;
        else R ← R ∪ {p̄} ;
        Γ ← ∅;
return R;

Function MARGINAL(v, M, G, w)
    return max(0, w(v) − Σ_{C∈M|C⊆N(v)} m(C));
Function INSERT(v, M, G, w)
    foreach C ∈ M | C ⊆ N(v) do C ← C ∪ {v} ;
```

then every neighbor u of v must be in $\mathcal{I}$. Therefore, we can move v from $\mathcal{I}$ to U, which decreases $w(\mathcal{I})$ by $w(v)$ and adds $w(v)$ copies of the clique $\{v\}$ to $\mathcal{M}$, hence leaving the value of $ub = w(\mathcal{I}) + |\mathcal{M}|$ unchanged. It is unsafe, however, to remove both x_v and $\overline{x_u}$ if $u \in N(v)$, since in this case $w(\mathcal{I})$ decreases by $w(v)$, but both u and v can be moved to $\mathcal{M}$ and therefore its cardinality might increase by more than $w(v)$.

Algorithm 4 is given the current assignment A, the clique multicover $\mathcal{M}$, and the slack $L = lb - w(I)$ on the cardinality of $\mathcal{M}$ with respect to 4. Moreover, we suppose that we can check if $p \in A$ is a *neighborhood literal*, that is, a direct consequence x_v of $\overline{x_u}$, i.e., of adding a neighbor $u \in N(v)$ in $\mathcal{I}$. It explores every literal (deductions and decision alike) in reverse chronological order. When the literal p is positive but not a neighborhood literal (line 2), we check the marginal cost on the bound to move the corresponding vertex $v(p)$ to the multicover (function MARGINAL). If it would invalidate equation 4, then $\overline{p}$ is added to the explanation R, otherwise it is inserted into $\mathcal{M}$ (function INSERT). When p is a neighborhood literal (line 1), then observe that it is so because of the next negative literal to be explored. We store such literals in a set Γ. Finally, when the literal p is negative we compute a marginal cost of inserting those of its neighbors that are not implied by an earlier decision, that is, the vertices in Γ. This cost is "optimistic" because for efficiency reasons we do not actually do any insertion in function MARGINAL, and therefore non-edges between vertices in Γ are not taken into account. If even this optimistic cost is too high, then we insert $\overline{p}$ into the explanation R, otherwise we try to insert as many of the corresponding vertices into $\mathcal{M}$ as possible, and the rest of the literals into the explanation.

For every clique in the multicover, we use a bitset to hold all the vertices that can possibly be inserted into that clique. Checking that the neighborhood of a vertex v includes a clique therefore costs $O(1)$. However, this set must be updated after each insertion by intersecting it with $N(v)$ hence every actual insertion of a vertex into a clique costs one bitset operation, i.e., $O(|V|)$. Function MARGINAL is called $O(|V|)$ times and costs $O(|V|)$ since it does not update any clique. Moreover, recall that every clique contains a vertex that appears in no other clique, and that vertex must be adjacent to v for it to be inserted in the clique. Therefore, v can be inserted in at most $O(|N(v)|)$ cliques. The overall number of insertions made by function INSERT is thus $O(|E|)$. EXPLAINBOUND therefore requires $O(|E|)$ bitset operations hence its overall complexity is in $O(|E||V|)$.

We can use the same method to explain bound-based pruning. To explain the literal $v \in \mathcal{V}$, we can consider the clique multicover $\mathcal{M}'$ described in section 5 in the graph induced by $V \setminus \{v\}$ and then explain the obtained lower bound. In practice, however, we found this method too expensive to apply for each pruning, hence we use the simpler PRUNE$_{\mathcal{V}}$ procedure, which only greedily removes vertices from the explanation as long as removing them from the dual bound computed for v is not worse than the primal bound.

Similarly, we use a naive explanation for the dominance rule: to explain that v is dominated, we use the set of assignments of its neighbors in the original graph that do not belong to the residual graph. This is done by the PRUNE$_{\mathcal{I}}$ procedure.

7 Primal Heuristic

We can often use information from the dual bound computation to find new solutions. We first note that the heuristic CLIQUECOVER in Algorithm 2 has the property that in any clique cover that it produces, no two cliques may be merged to form a larger clique. Therefore, two vertices assigned to singleton cliques cannot be neighbours.

Based on this observation, let $\mathcal{I}$ be the current independent set and $\mathcal{I}^{\mathcal{M}}$ the set of vertices assigned to singleton cliques in $\mathcal{M}$. Then $\mathcal{I} \cup \mathcal{I}^{\mathcal{M}}$ is also an independent set. If there exists a vertex $v \in V \setminus (\mathcal{I} \cup \mathcal{I}^{\mathcal{M}})$ such that $v \notin N(\mathcal{I}^{\mathcal{M}})$, then $\mathcal{I} \cup \mathcal{I}^{\mathcal{M}} \cup \{v\}$ is also an independent set. We pick one such vertex arbitrarily and iterate until we reach a fix point which we store if its cost is better than the current primal bound.

In the case of clique multicovers, each iteration of the algorithm that we described in section 4 can be used to generate and then greedily improve an independent set.

Example 5. *Consider again Figure 1 and the 3 clique covers computed in figures 1(a) – 1(c). The first clique cover* $\{\{a, b, c\}, \{d, e, f\}\}$ *contains no singleton cliques, hence we start with* $\mathcal{I} = \emptyset$*. We greedily add vertex* a *from the first clique and vertex* e *from the second, to get an independent set of weight* 13 *and a corresponding clique cover* $\{b, c, d, f\}$ *of weight* 30*. The second contains the singleton clique* $\{b\}$*. We greedily add* f *from the clique* $\{a, d, f\}$*, to get an independent set of weight 18 and a vertex cover* $\{a, c, d, e\}$ *of weight*

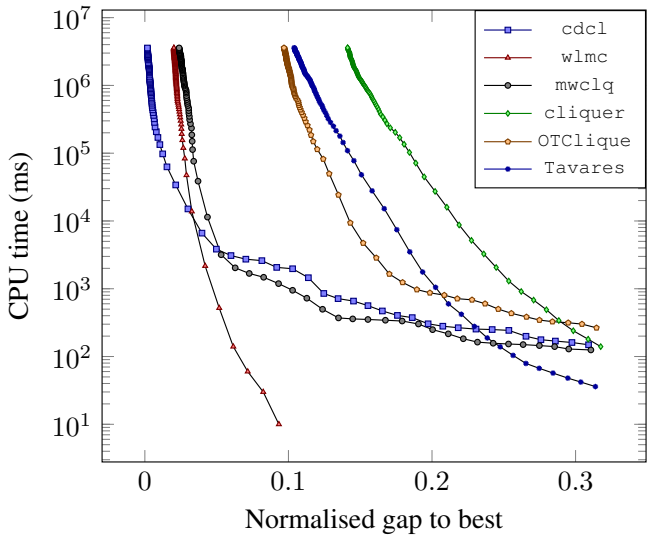


Figure 3: Normalised gap to best over time (all instances)

25. Since that matches the dual bound computed, this solution is optimal. The third clique cover yields the same solution.

8 Search Heuristic

As was observed in [McCreesh and Prosser, 2014], given a clique cover $\mathcal{C}$, since for every clique $C \in \mathcal{C}$ at most one vertex in C can be in the independent set, C can be seen as the domain of a variable that we can branch on. The well known heuristic that choses a vertex in the last clique in $\mathcal{C}$ can therefore be seen as similar to the *minimum domain* heuristic since for usual greedy procedures the last clique tends to be the smallest. Moreover, using the maximum degree of a vertex, or the maximum weight of a vertex are also natural criteria.

We found the following criterion to be the best: we branch first on the vertex v whose clique $C(v)$ in the first "layer" of the clique multicover is such that $|C(v)|/w(v)$ is minimum and put it the $\mathcal{I}$ by setting $\overline{x_v}$. Moreover, we observed that VSIDS [Moskewicz *et al.*, 2001] is efficient for finding good solutions quickly, although the criterion above tends to be best in the long run. Therefore, all the results that we report for our method in section 9 were obtained using VSIDS for up to 50000 nodes, and then switching to branching on v minimising $|C(v)|/w(v)$.

9 Experimental Evaluation

We implemented our approach in C++, denoted `cdcl`, on top of the MINICSP solver[2]. We compared it with the solvers `mwclq` [Fang *et al.*, 2016], `wlmc` [Jiang *et al.*, 2017], `cliquer` [Östergård, 2001], `OTClique` [Shimizu *et al.*, 2017] and an implementation of Tavares' method by the authors of [McCreesh *et al.*, 2017]. We used the benchmarks introduced in [McCreesh *et al.*, 2017] divided in four classes:

`WDP` Winner Determination Problem, where the graph represents the compatibility of item sets, weights represent bids, and the maximum clique stands for the solution of maximum profit for the auctioneer [Lau and Goh, 2002];

`REF` Research Excellence Network, where the clique stands for the optimal set of publications that a university department can provide to the authority assessing it (generated by [McCreesh *et al.*, 2017]);

`EC-CODE` Error-correcting Codes, where the clique stands for a set of words maximally pair-wise distant (instances due to [Östergård, 1999], reconstructed by [McCreesh *et al.*, 2017]);

`KIDNEY` Kidney Exchange, where the clique stands for a maximally desirable set of donor/patient exchanges. The instances were generated by [McCreesh *et al.*, 2017] using data from [Dickerson *et al.*, 2012] and a weighting scheme from [Manlove and O'malley, 2015].

Moreover, we also provide results on the classic `DIMACS` and `BHOSLIB` benchmarks. Since those graphs are not weighted, we used the weight function $w(v) = (v \mod 200) + 1$ with vertices numbered from 1 to n, a standard practice in evaluating maximum weighted clique algorithms, attributed to Pullan [Pullan, 2008]. The number of instances in each class is shown in the 2nd column of Table 1 (omitting 6 trivial or empty `KIDNEY` instances). For DIMACS, we use two copies of each instance to get a total of 160 instances instead of the 80 in that set. The second copy of each instance is randomly shuffled so as to avoid reveal any bias in the algorithms. These shuffled instances were generated by [McCreesh *et al.*, 2017].

Every method was run with a time limit of 1h and a memory limit of 3.5GB[3] on a cluster with 4 nodes, each with 35 Intel Xeon E5-2695 2.10GHz cores running Ubuntu Linux 16.04.4.

For each class of benchmarks, we compute the geometric mean of the objective, that is, the maximum weight of the clique found. Moreover, we compute the *mean normalised gap to the best solution*. On an instance where the best solution found by the best (resp. worst) method has weight u (resp. l), the normalised gap $g(w)$ of a clique of weight w is:

$$g(w) = \begin{cases} 0 & \text{if } u = l \\ (u - w)/(u - l) & \text{otherwise} \end{cases}$$

A mean normalised gap of 0 (resp. 1) therefore indicates that the method systematically finds the solution with best (resp. worst) objective value. The geometric average is quite robust to outliers, in particular more so than the arithmetic average. The normalised gap to the best solution is even more so, allowing to aggregate extremely heterogeneous results. We show in Figure 3 the evolution of the normalised gap over time for the whole data set. We can see that other solvers initially find good solutions faster (the figure is truncated). However, the quality of the solutions found by `cdcl` increases much more rapidly and after less than one second it is already better than all methods except `mwclq` and `wlmc`. Morever, both reach a plateau earlier and `cdcl` is the best overall.

[2]Sources available at: https://bitbucket.org/gkatsi/minicsp

[3]Only mwclq exceeded the memory limit, on 7 instances.

		cdcl		wlmc		mwclq		cliquer		OTClique		Tavares	
		objective	gap	objective	gap	objective	gap	objective	gap	objective	gap	objective	gap
BHOSLIB	(40)	4671.36	0.010	3770.83	0.225	4598.76	0.025	835.05	0.975	1619.57	0.623	4277.46	0.109
WDP	(50)	84.84**M**	0.145	85.53**M**	0.000	85.53**M**	0.000	85.53**M**	0.000	85.53**M**	0.000	84.81**M**	0.124
EC-CODE	(15)	97.31	0.000	97.31	0.000	96.88	0.067	97.31	0.000	97.31	0.000	97.31	0.000
DIMACS	(160)	3277.00	0.013	3232.41	0.044	3252.04	0.016	2079.63	0.320	2496.57	0.183	3146.91	0.096
REF	(129)	129.82	0.000	128.11	0.019	128.61	0.014	105.06	0.186	117.88	0.108	129.24	0.007
KIDNEY	(188)	549.71**B**	0.000	549.41**B**	0.001	516.48**B**	0.208	537.69**B**	0.111	540.15**B**	0.060	544.41**B**	0.029

Table 1: Comparison with the state of the art: geometric mean clique weight (objective) & mean normalised gap to best (gap)

		cdcl		cdcl\pru		cdcl\dom		cdcl\learn		cdcl\prim		$\text{cdcl}_\emptyset$	
		objective	gap	objective	gap	objective	gap	objective	gap	objective	gap	objective	gap
BHOSLIB	(40)	4671.36	0.010	4671.06	0.013	4636.09	0.018	4417.32	0.090	4673.67	0.013	4302.29	0.134
WDP	(50)	84.84**M**	0.145	84.40**M**	0.369	84.71**M**	0.184	85.45**M**	0.016	84.79**M**	0.172	85.32**M**	0.032
EC-CODE	(15)	97.31	0.000	97.31	0.000	96.49	0.067	97.31	0.000	96.49	0.067	97.31	0.000
DIMACS	(160)	3277.00	0.013	3267.03	0.059	3270.43	0.021	3231.94	0.046	3276.68	0.017	3188.82	0.127
REF	(129)	129.82	0.000	129.82	0.000	129.83	0.000	129.06	0.009	129.82	0.000	128.41	0.016
KIDNEY	(188)	549.71**B**	0.000	549.71**B**	0.000	549.71**B**	0.000	549.56**B**	0.001	549.71**B**	0.000	545.28**B**	0.024

Table 2: Factor analysis: geometric mean clique weight (objective) & mean normalised gap to best (gap)

We report the results for each class of instances in Table 1 where, for each class and each metric, we highlight the best method.[4] Although no method strictly dominates the other on the whole data set, cdcl finds the best solutions on 4 out of the 5 classes of instances. On DIMACS and BHOSLIB, it finds the best solutions in average, however the normalised gap is not null, indicating that other methods (mainly mwclq) are able to find better solutions in some cases.

On WDP, cdcl is the second worst method. Interestingly, it appears that clause learning does not help and even hinders the performance on these instances (see Table 2). This is not clear whether this is simply due to the time overhead of learning, which can be significant, or if the clauses we learn are somehow counterproductive in this context.

On EC-CODE, cdcl finds the same best solutions as wlmc, cliquer and OTClique. However, it explores about as large a search tree as wlmc, while requiring 15 times more CPU time per node. The search trees explored by cliquer and OTClique are orders of magnitude larger, but they explore about 500 times more nodes per second. As a result, these three methods solve every EC-CODE instance to optimality while cdcl proves only 2/3 of the instances.

On REF and KIDNEY, on the other hand, cdcl is clearly the best method. It finds the heaviest cliques in every single instance, as shown by the normalised gap of 0. Moreover, it proves 92.6% of the KIDNEY instances whereas the second best solver, wlmc, can only prove 78.7%. On REF, however, these ratios are in favour of wlmc: 82.2% vs. 81.4%.

We report in Table 2 further tests to assess the impact of the different contributions. We denote cdcl\pru, cdcl\dom, cdcl\learn and cdcl\prim the versions of cdcl where pruning, dominance, learning and the primal bound are turned off, respectively. Each factor has a tradeoff that can be positive or negative according to the data set. The impact of the primal heuristic is rather slight, but does not incur a significant overhead. On the other hand, clause learning, pruning and to a lesser extent dominance entail a slow down of up to 100%, which explains why in some cases better solutions can be found when turning them off. The version where every factor is turned off ($\text{cdcl}_\emptyset$), however, is clearly the worst.

[4] Results on individual instances are available here: http://homepages.laas.fr/ehebrard/mwclq.pdf

10 Conclusions

We have introduced the basis of a conflict-driven clause learning approach to the minimum weight vertex cover problem. This approach uses an effective implementation of the pruning rule based on marginal costs of Babel's clique multicover dual bound, and a novel and effective dominance rule. Moreover, we described a non-trivial clause minimization algorithm to compute explanations of failures due to this bound.

Together with an opportunistic primal heuristic and a search heuristic taking advantage of domain knowledge as well clause activity, these techniques are shown to be competitive with the state of the art on a large data set. The experimental results show that it clearly outperforms the best known methods on several classes of problems.

References

[Babel, 1994] Luitpold Babel. A Fast Algorithm for the Maximum Weight Clique Problem. *Computing*, 52(1):31–38, 1994.

[Bomze *et al.*, 1999] Immanuel M. Bomze, Marco Budinich, Panos M. Pardalos, and Marcello Pelillo. The Maximum

Clique Problem. In *Handbook of Combinatorial Optimization*, pages 1–74. Kluwer Academic Publishers, 1999.

[Brélaz, 1979] Daniel Brélaz. New Methods to Color the Vertices of a Graph. *Commun. ACM*, 22(4):251–256, 1979.

[Day and Sankoff, 1986] William H. E. Day and David Sankoff. Computational Complexity of Inferring Phylogenies by Compatibility. *Systematic Zoology*, 35(2):224–229, 1986.

[Dickerson *et al.*, 2012] John P Dickerson, Ariel D Procaccia, and Tuomas Sandholm. Optimizing kidney exchange with transplant chains: Theory and reality. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*, pages 711–718. International Foundation for Autonomous Agents and Multiagent Systems, 2012.

[Fang *et al.*, 2016] Zhiwen Fang, Chu-Min Li, and Ke Xu. An Exact Algorithm Based on MaxSAT Reasoning for the Maximum Weight Clique Problem. *J. Artif. Int. Res.*, 55(1):799–833, 2016.

[Jiang *et al.*, 2017] Hua Jiang, Chu-Min Li, and Felip Manyà. An exact algorithm for the maximum weight clique problem in large graphs. In *Proceedings of the 31st Conference on Artificial Intelligence (AAAI)*, pages 830–838, 2017.

[Jiang *et al.*, 2018] Hua Jiang, Chu-Min Li, Yanli Liu, and Felip Manyà. A two-stage maxsat reasoning approach for the maximum weight clique problem. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence (AAAI-2018)*, 2018.

[Katsirelos and Bacchus, 2005] George Katsirelos and Fahiem Bacchus. Generalized nogoods in CSPs. In *Proceedings of the 20th National Conference on Artificial Intelligence (AAAI)*, 2005.

[Lau and Goh, 2002] Hoong Chuin Lau and Yam Guan Goh. An intelligent brokering system to support multi-agent web-based 4/sup th/-party logistics. In *Proceedings of the 14th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2002)*, pages 154–161. IEEE, 2002.

[Lick and White, 1970] Don R. Lick and Arthur T White. k-degenerate graphs. *Canadian Journal of Mathematics*, 22:1082–1096, 1970. doi:10.4153/CJM-1970-125-1.

[MacWilliams and Sloane, 1978] Florence J. MacWilliams and Neil J.A. Sloane. *The Theory of Error-Correcting Codes*. North-holland Publishing Company, 2nd edition, 1978.

[Manlove and O'malley, 2015] David F. Manlove and Gregg O'malley. Paired and altruistic kidney donation in the uk: Algorithms and experimentation. *J. Exp. Algorithmics*, 19:2.6:1–2.6:21, January 2015.

[Marques Silva and Sakallah, 1999] Joao P. Marques Silva and Karem A. Sakallah. GRASP—a search algorithm for propositional satisfiability. *IEEE Transactions on Computers*, 48(5):506–521, May 1999.

[McCreesh and Prosser, 2014] Ciaran McCreesh and Patrick Prosser. Reducing the Branching in a Branch and Bound Algorithm for the Maximum Clique Problem. In *Proceedings of the 20th International Conference on Principles and Practice of Constraint Programming (CP)*, pages 549–563, 2014.

[McCreesh *et al.*, 2017] Ciaran McCreesh, Patrick Prosser, Kyle Simpson, and James Trimble. On Maximum Weight Clique Algorithms, and How They Are Evaluated. In *Proceedings of the 23rd International Conference on the Principles and Practice of Constraint Programming (CP)*, pages 206–225, 2017.

[Mitchell, 2005] David G. Mitchell. A SAT solver primer. *EATCS Bulletin*, 85:112–132, 2005.

[Moskewicz *et al.*, 2001] Matthiew Moskewicz, Conor Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an efficient SAT solver. In *Proceedings of the 39th Design Automation Conference (DAC)*, pages 530–535, 2001.

[Östergård, 2001] Patric R. J. Östergård. A New Algorithm for the Maximum-weight Clique Problem. *Nordic J. of Computing*, 8(4):424–436, 2001.

[Östergård, 2002] Patric R. J. Östergård. A fast algorithm for the maximum clique problem. *Discrete Applied Mathematics*, 120(1–3):197 – 207, 2002. Special Issue devoted to the 6th Twente Workshop on Graphs and Combinatorial Optimization.

[Pullan, 2008] Wayne Pullan. Approximating the maximum vertex/edge weighted clique using local search. *Journal of Heuristics*, 14(2):117–134, Apr 2008.

[Samudrala and Moult, 1998] Ram Samudrala and John Moult. A Graph-theoretic Algorithm for Comparative Modeling of Protein Structure. *Journal of Molecular Biology*, 279(1):287–302, 5 1998.

[Shimizu *et al.*, 2017] Satoshi Shimizu, Kazuaki Yamaguchi, Toshiki Saitoh, and Sumio Masuda. Fast maximum weight clique extraction algorithm: Optimal tables for branch-and-bound. *Discrete Applied Mathematics*, 223:120–134, 2017.

[Strickland *et al.*, 2005] Dawn M. Strickland, Earl Barnes, and Joel S. Sokol. Optimal Protein Structure Alignment Using Maximum Cliques. *Operations Research*, 53(3):389–402, 2005.

[Tavares, 2016] Wladimir Araujo Tavares. *Algoritmos exatos para problema da clique maxima ponderada. (Exact algorithms for the maximum-weight clique problem / Algorithmes pour le problème de la clique de poids maximum)*. PhD thesis, University of Avignon, France, 2016.

[Östergård, 1999] Patric R.J. Östergård. A new algorithm for the maximum-weight clique problem. *Electronic Notes in Discrete Mathematics*, 3:153 – 156, 1999. 6th Twente Workshop on Graphs and Combinatorial Optimization.