ULE-MWC: An UnLocking-Enhanced Lookahead Framework for Exact Maximum Weight Clique Search

Mingming Jin[1] , **Chu-Min Li**[2] , **Kun He**[1*]

[1]School of Computer Science and Technology, Huazhong University of Science and Technology, China
[2]MIS,Université de Picardie Jules Verne, France
brooklet60@hust.edu.cn

Abstract

The Maximum Weight Clique Problem (MWCP) is NP-hard and is typically solved exactly within a Branch-and-Bound (BnB) framework, where the quality of upper bounds critically determines the pruning efficiency. Recent solvers enhance independent-set-based bounds using MaxSAT reasoning. However, their effectiveness is limited by two inherent limitations: (i) independent sets in conflicts are discarded after a single use, preventing their reuse in subsequent reasoning; and (ii) conflicts are detected in a fixed order regardless of their potential contribution to bound tightening. To address these issues, we propose two complementary techniques. The UnLocking Mechanism (ULM) preserves conflict cores and enables the reuse of independent sets in subsequent reasoning, while the Benefit-Enhanced Strategy (BES) steers the MaxSAT reasoning process toward conflicts and independent sets with higher pruning potential. By integrating ULM and BES, we develop a new solver, ULE-MWC (UnLocking Enhanced MWC). Experimental results on standard benchmarks show that ULE-MWC consistently yields tighter upper bounds and outperforms state-of-the-art exact solvers in both runtime and search efficiency.

1 Introduction

Given an undirected simple graph $G = (V, E)$, where V denotes the vertex set and E is the edge set, a *clique* is a subset of vertices such that every pair of vertices is adjacent in G. The Maximum Clique Problem (MCP) aims to find a clique of maximum size in G. The Maximum Weight Clique Problem (MWCP) extends MCP by assigning each vertex v a non-negative weight $w(v)$, and seeks a clique C that maximizes the total weight $w(C) = \sum_{v \in C} w(v)$. In this paper, we focus on algorithms for solving MWCP.

MCP is a classical NP-hard problem [Hartmanis, 1982]. Since MCP is a special case of MWCP where all vertex weights are equal to 1, MWCP is also NP-hard. Due to its theoretical importance and practical value, MWCP has been applied in many areas, including coding theory [Zhian *et al.*, 2013], computer vision [Zhang *et al.*, 2014; Sanroma *et al.*, 2016], protein structure prediction [Mascia *et al.*, 2010], and genomics [Butenko and Wilhelm, 2006].

A large range of algorithms have been proposed for MWCP, which are generally divided into two categories: heuristics and exact methods. The most efficient heuristic algorithms include MN/TS [Wu *et al.*, 2012], FastWClq [Cai and Lin, 2016], RRWL [Fan *et al.*, 2017], ReTS [Zhou *et al.*, 2017], and SCCWalk [Wang *et al.*, 2020]. And representative exact methods include Cliquer [Östergård, 2002], MWCLQ [Fang *et al.*, 2016], WLMC [Jiang *et al.*, 2017], TSM-MWC [Jiang *et al.*, 2018], and BBMWC [San Segundo *et al.*, 2019], most of which are based on the Branch-and-Bound (BnB) framework.

A BnB algorithm for MWCP usually maintains a growing partial clique S and a corresponding candidate set P. Vertices in P are adjacent to all vertices in S, so adding any vertex from P to S keeps S a clique. Let $\omega(G, w)$ denote the weight of the best clique found so far in G. If an upper bound on the maximum weight clique that can be obtained on the current search node is no greater than $\omega(G, w)$, the branch can be safely pruned. Therefore, the quality of the upper bounds plays a crucial role in reducing the search space and improving the efficiency of the algorithms.

MaxSAT reasoning is an effective way to improve the upper bounds. It was developed for MCP [Li and Quan, 2010; Li *et al.*, 2017; Li *et al.*, 2018], and has recently been extended to MWCP [Jiang *et al.*, 2017; Jiang *et al.*, 2018]. Starting from an independent set based upper bound, MaxSAT reasoning formulates the clique problem as a partial MaxSAT instance and further improves the bound by identifying conflicts, *i.e.*, groups of independent sets whose vertices cannot be selected to form a clique simultaneously.

Though MWCP solvers based on MaxSAT reasoning have achieved strong pruning performance, there are still several limitations. Once a conflict is identified, existing methods compute its conflicting weight, split the involved independent sets, and permanently remove the conflicting parts, which prevents the reuse of potentially valuable independent sets in later reasoning. Besides, conflicts are encountered in a fixed order, and the bound reduction is dominated by low-weight independent sets in it. Thus, a conflict often provides

*Corresponding author.

Algorithm 1: BnB(G, P, O, S, C^*)

Input: $G = (V, E, w)$, a vertex ordering O over V, the growing clique S, the candidate set P, and the maximum weight clique C^* found so far.
Output: The maximum weight clique C^*

```
if |P| = 0 then return S;
B ← GetBranches(G[P], w(C*) − w(S), O);
if B = ∅ then return C*;
Let B = {b_1, b_2, ··· , b_|B|} w.r.t. O and A ← P \ B;
for i = |B| to 1 do
    P' = N(b_i) ∩ ({b_{i+1}, ··· , b_|B|} ∪ A);
    C' ← BnB(G, P', O, S ∪ {b_i}, C*);
    if w(C') > w(C*) then C* ← C';
return C*;
```

minor bound reductions while still requiring the full cost of MaxSAT reasoning. Consequently, the pruning gain is not always proportional to the computational effort.

To address these limitations, we introduce two complementary strategies. The UnLocking Mechanism (ULM) preserves conflict cores by temporarily locking them, which allows the associated independent sets to participate in later reasoning. The Benefit-Enhanced Strategy (BES) guides the reasoning process by filtering independent sets to detect conflicts with higher pruning potential. We introduce ULM and BES into the MaxSAT reasoning process and develop the ULE-MWC solver. Extensive experiment results show that ULE-MWC consistently produces tighter bounds and achieves faster solving performance than state-of-the-art MWCP solvers on standard benchmarks.

2 Preliminaries

Given an undirected simple graph $G = (V, E, w)$, where $V = \{v_1, v_2, \ldots, v_n\}$ is the vertex set, E is the edge set, and $w : V \to \mathbb{R}_{\geq 0}$ assigns a non-negative weight to each vertex. The density of G is defined as $2|E|/(|V|(|V|-1))$. For each vertex $v_i \in V$, we define $w(v_i)$ as the weight of it, and we also use $v_i^{w(v_i)}$ to represent vertex v_i together with its weight $w(v_i)$. The neighbor set of v_i is denoted as $\mathcal{N}(v_i)$. For any clique $C \subseteq V$, its total weight is defined as $w(C) = \sum_{v_i \in C} w(v_i)$.

A weight independent set $I \subseteq V$ is a set of pairwise non-adjacent vertices. The weight of I is defined as the maximum weight in I, *i.e.* $w^*(I) = max_{v_i \in I}(w(v_i))$. Let $I = \{v_1^{w(v_1)}, v_2^{w(v_2)}, \ldots, v_r^{w(v_r)}\}$ be a weight independent set whose vertices are sorted in non-increasing order of weights, *i.e.*, $w(v_1) \geq \cdots \geq w(v_k) > \beta \geq w(v_{k+1}) \geq \cdots \geq w(v_r)$. The *split* operation *split*(I, β) divides I into two subsets: $I' = \{v_1^{\beta}, v_2^{\beta}, \cdots, v_k^{\beta}, v_{k+1}^{w(v_{k+1})}, \cdots, v_r^{w(v_r)}\}$ and $I'' = \{v_1^{w(v_1)-\beta}, v_2^{w(v_2)-\beta}, \cdots, v_k^{w(v_k)-\beta}\}$.

2.1 Basic BnB Framework for MWCP

Algorithm 1 outlines a basic BnB framework for MWCP. Given a weight graph $G = (V, E, w)$, a candidate set P, a vertex ordering O and a partial clique S, the algorithm maintains the best clique C^* found so far. At each search node, it invokes *GetBranches* to divide P into two subsets, A and B. The vertices in A induce a subgraph whose maximum weight clique is bounded by $w(C^*) - w(S)$, and can therefore be pruned safely. The remaining vertices form the branching set $B = P \setminus A = \{b_1, \ldots, b_{|B|}\}$. If B is empty, the current search branch is pruned, because the clique C can not grow to a clique of weight greater than $w(C^*)$. Otherwise, the algorithm branches on each vertex in B, from $b_{|B|}$ to b_1, to search for a clique containing b_i in the induced subgraph by $\mathcal{N}(b_i) \cap (\{b_{i+1}, \cdots, b_{|B|}\} \cup A)$. Whenever a clique with larger weight is found, C^* is updated accordingly.

The pruning effectiveness of the BnB framework largely depends on how *GetBranches* divides the candidate set. A good partition moves as many vertices as possible into A without violating the upper-bound condition, leaving fewer vertices in B to branch on. TSM-MWC [Jiang *et al.*, 2018] follows this idea by adapting a two-stage MaxSAT reasoning inside *GetBranches*, which leads to tighter upper bounds and a smaller search tree.

2.2 MaxSAT Reasoning for MWCP

Before MaxSAT reasoning, TSM-MWC divides the candidate set P into two subsets by partitioning weight independent sets. Vertices covered by a set of independent sets $\Pi = \{I_1, \ldots, I_k\}$ are grouped into A, such that $\sum_{I_i \in \Pi} w^*(I_i) \leq w(C^*) - w(S)$, and all other vertices form B. A vertex may appear in multiple independent sets in Π, and its weight is distributed among these sets while keeping the total weight unchanged, *i.e.*, $w(v) = \sum_{I_i \in \Pi,\, v \in I_i} w_i(v)$, where $w_i(v)$ denotes the portion of v's weight assigned to I_i.

A *conflict* is a set of independent sets, from which no clique can choose one vertex from each set. MaxSAT reasoning detects such conflicts through unit propagation (UP) to tighten the upper bound. During UP, the algorithm keeps track of the independent sets that contain only one vertex and removes all non-neighbors of this vertex from the other sets. If an independent set I_e becomes empty, a conflict is detected, consisting of I_e and the independent sets that caused all its vertices to be removed, and the upper bound is decreased. Vertices in B are tentatively added to A in the reverse order, and conflicts in the expanded set are analyzed. If the resulting upper bound is no greater than $w(C^*) - w(S)$, the vertex can be safely pruned.

More concretely, let Π denote the current independent sets. When a vertex from B is tentatively added to A, the resulting upper bound may violate the constraint, *i.e.* $ub > w(C^*) - w(S)$. In that case, MaxSAT reasoning searches for a conflict $F = \{I_1, I_2, \ldots, I_k\} \subseteq \Pi$. Once such a conflict is found, it computes a shared weight $\beta = \min(w^*(I_1), w^*(I_2), \ldots, w^*(I_k))$. Each independent set $I_i \in F$ is then split into two parts, I_i' and I_i'', using *split*(I_i, β). The sets I_i' all have weight β and together form a simplified conflict. These conflict sets are simply removed from further reasoning. The remaining parts, $\Pi_2 = \{I_i'' | w^*(I_i'') > 0\}$, are kept for subsequent reasoning. As a result, Π is updated to $(\Pi \setminus F) \cup \Pi_2$, and the upper bound of A decreases from ub to $ub - \beta$.

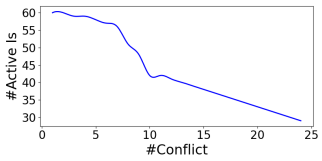

Figure 1: Decrease of the number of Independent Sets During MaxSAT Reasoning

3 ULE-MWC: An Efficient BnB Algorithm With an Unlocking-Enhanced Mechanism

In this section, we analyze the limitations of existing MaxSAT reasoning for MWCP. To overcome these limitations, we introduce two complementary strategies, the UnLocking Mechanism (ULM) and the Benefit-Enhanced Strategy (BES). By incorporating both strategies into the MaxSAT reasoning, we develop the enhanced BnB solver ULE-MWC.

3.1 Limitations of Existing MaxSAT Reasoning

Although MaxSAT reasoning is effective for tightening upper bounds in MWCP, its current design has clear limitations.

Limitation 1: Irreversible shrinkage of the reasoning pool. In MaxSAT reasoning, once a conflict is detected, the independent sets involved are split, and the conflicting parts are removed from the subsequent reasoning process.

Example 1. *Assume four independent sets I_1, I_2, I_3, and I_4, with weights $w^*(I_1) = 1$ and $w^*(I_2) = w^*(I_3) = w^*(I_4) = 10$. A conflict $F = \{I_1, I_2, I_3\}$ is detected, with a minimum weight of 1. After splitting and removing the conflicting parts $\{I_1, I_2', I_3'\}$, three independent sets left: I_2'' and I_3'' with weight 9, and I_4 with weight 10.*

In the early stages, this approach is often very efficient to reduce the upper bound, as the number of independent sets is relatively large. However, as the reasoning process continues, more conflicting independent sets are removed, and the pool of available independent sets gradually shrinks, making it harder to find new conflicts.

Figure 1 provides evidence of this limitation, obtained from TSM-MWC when solving benchmark instances. As more conflicts are detected, the number of independent sets steadily decreases, which directly limits further reasoning.

Limitation 2: Conflict detection without benefit awareness. Another limitation lies in conflict detection. Since independent sets are processed in a fixed order, conflict identification is highly order-dependent. Moreover, the reduction of the upper bound is governed by the smallest weight among the independent sets in the conflict. As a result, many conflicts involve low-weight independent sets, leading to small bound reductions while still costing the same reasoning effort as more useful conflicts. As a result, much time is spent on conflicts with little pruning effect.

Taken together, these limitations show that existing MaxSAT reasoning neither reuses conflicting independent sets nor focuses reasoning effort on the most effective conflicts. This motivates our two strategies: an UnLocking Mechanism to reuse conflict cores, and a Benefit-Enhanced Strategy to prioritize conflicts with higher pruning potential.

Algorithm 2: AddCore($F, \Pi, \mathcal{C}, \mathcal{L}$)

Input: A conflict F, the set of independent sets Π, conflict cores $\mathcal{C}$, locked independent sets $\mathcal{L}$.
Output: The updated Π, $\mathcal{C}$ and $\mathcal{L}$ and the reduced weight β.

1 $\beta = \min_{I_i \in F} w^*(I_i)$;
2 $\Pi' \leftarrow \emptyset$, $c \leftarrow \emptyset$;
3 $c.w \leftarrow \beta$, $c.uw \leftarrow 0$, $c.active = true$;
4 **for** $I_i \in F$ **do**
5 $\quad (I_i', I_i'') \leftarrow split(I_i, \beta)$;
6 $\quad \Pi' \leftarrow \Pi' \cup \{I_i''\}$;
7 $\quad \mathcal{L} \leftarrow \mathcal{L} \cup \{I_i'\}$, $I_i'.unlock = false$;
8 $\quad I_i'.c = c$, $c \leftarrow c \cup \{I_i'\}$;
9 $\quad$ **if** $I_i \in \mathcal{L}$ *and* $(c' = I_i.c,\ c'.active = true)$ **then**
10 $\quad\quad c'.active = false$, $c.w \leftarrow c.w + c'.w$;
11 $\quad\quad$ **for** $I_j \in c'$ **do**
12 $\quad\quad\quad c \leftarrow c \cup \{I_j\}$, $I_j.c = c$;
13 $\Pi \leftarrow (\Pi \setminus F) \cup \Pi'$, $\mathcal{C} \leftarrow \mathcal{C} \cup \{c\}$;
14 **return** $(\Pi,\ \mathcal{C},\ \mathcal{L},\ \beta)$;

3.2 Lookahead Through Unlocking Mechanism

To enable the reuse of independent sets involved in conflicts, we introduce *conflict cores* to maintain the conflicts.

Definition 1 (Conflict Core)**.** *A* conflict core *c is a set of independent sets with a non-negative penalty value $c.w$. Let $w_t(c) = \sum_{I \in c} w^*(I)$ denote the total weights of the independent sets in c, and $V(c)$ be the set of vertices with corresponding weights contained in these independent sets. The core satisfies: (core invariant) For any clique $C \subseteq V(c)$, $w(C) \leq w_t(c) - c.w$.*

During MaxSAT reasoning, independent sets may become empty due to propagation. Let $c.uw$ denote the total weight of independent sets in c that have become empty. When $c.uw \geq c.w$, the penalty enforced by the core is fully paid, and the Core Invariant is satisfied independently of the remaining independent sets. Hence, all remaining locked independent sets in c can be safely unlocked and reactivated without affecting the correctness of the upper bound.

Algorithm 2 describes how a conflict core is constructed from a detected conflict F. It first computes the minimum shared weight β among the independent sets in F (line 1) and initializes a new conflict core (lines 2–3). Each independent set in F is then split by β (line 5). The part with weight β is added to the new conflict core and locked to prevent immediate reuse (lines 7–8), while the remaining parts with positive weight are kept for reasoning (line 6). If the locked part originates from an existing active conflict core, that core is deactivated and merged into the new one, preserving previously derived conflict information (lines 9–12). Finally, the algorithm updates the pool of independent sets and the set of conflict cores (line 13).

Algorithm 3 extends basic MaxSAT reasoning by introducing conflict cores, so that independent sets involved in conflicts are not permanently removed but can be reused in subsequent reasoning. The algorithm incrementally tightens δ,

Algorithm 3: ULM-MaxSAT($\Pi, \mathcal{C}, \mathcal{L}, t, ub, \delta$)

Input: A set of independent sets Π, conflict cores $\mathcal{C}$, locked independent sets $\mathcal{L}$, integers t, ub and δ.
Output: Updated δ.

1 **while** $\exists$ *unit* $I_u\{v\} \in \Pi$ *or unlocked* $L_u\{v\} \in \mathcal{L}$ **do**
2 Remove non-neighbors of v from $I \in \Pi$ and $L \in \mathcal{L}$;
3 **if** $\exists$ *empty* $I_e \in \Pi$ *or unlocked* $L_e \in \mathcal{L}$ **then**
4 $F \leftarrow$ collect Is and Ls responsible of removing all vertices from I_e or L_e;
5 Restore removed vertices and conflict cores;
6 $(\Pi, \mathcal{C}, \mathcal{L}, \beta) \leftarrow AddCore(F, \Pi, \mathcal{C}, \mathcal{L})$;
7 $\delta \leftarrow \delta - \beta$;
8 **if** $ub + \delta \leq t$ **then break**;
9 **if** $\exists$ *empty* $L_e \in \mathcal{L}$, *and* $L_e.unlock = false$ **then**
10 $c \leftarrow L_e.c$, $c.uw \leftarrow c.uw + w(L_e)$;
11 **if** $c.uw \geq c.w$ **then**
12 **for** $L \in c$ **do**
13 $L.unlock = true$;
14 **return** δ;

which denotes the increase in the upper bound caused by tentatively adding the current branching vertex into A. Here, ub is the upper bound of the weight clique induced by the original A, and $t = w(C^*) - w(S)$ is the pruning threshold. The branch can be pruned once $ub + \delta \leq t$.

The procedure repeatedly performs propagation as long as there exists a unit independent set or an independent set in an unlocked conflict core (line 1). When a unit set $\{v\}$ is found, all vertices non-adjacent to v are removed from all other independent sets (line 2). If this operation makes an independent set or an unlocked set empty, a conflict is detected (line 3). The independent sets responsible for this conflict are then collected (line 4), and a new conflict core is constructed by *AddCore*, which decreases δ by the extracted shared weight (lines 6–7). After each detected conflict, all previously removed vertices and all active conflict cores are restored (line 5). If $ub + \delta \leq t$, the reasoning stops (line 8).

For an empty locked independent set whose conflict core is still locked, its weight is accumulated into the corresponding core (lines 9–10). Once the accumulated weight reaches the penalty of the core, the Core Invariant is satisfied, and all remaining locked independent sets in the core are unlocked and reactivated for further reasoning (lines 11–13). The algorithm terminates when no further propagation is possible and returns the final value of δ (line 14).

Proposition 1 establishes the soundness of Algorithm 2.

Proposition 1. *Every conflict core generated by Algorithm 2 satisfies the Core Invariant in Definition 1.*

Proof. Let $F = \{I_1, \cdots, I_k\}$ be the detected conflict, $\beta = min_{I_i \in F}\ w^*(I_i)$. By definition of conflict, no clique contained in $\bigcup_{I_i \in F} I_i$ can simultaneously select vertices from all sets in F. Hence, any clique $C \subseteq V(c)$ must lose at least β total weight relative to $\sum_{i=1}^{k} w^*(I_i)$.

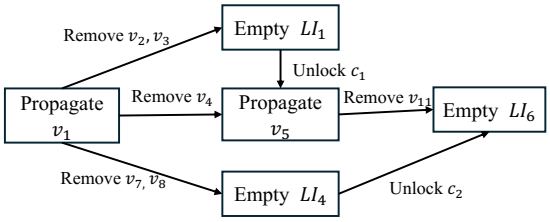

Figure 2: Reasoning process of example 2

.

Case 1: The conflict core does not involve any independent set from existing cores. Algorithm 2 constructs a new conflict core c by locking weight β from each $I_i \in F$, and sets $c.w = \beta$. For any clique $C \subseteq V(c)$, since no other conflict is involved, the above argument applies, $w(C) \leq \sum_{i=1}^{k} w^*(I_i) - \beta = w_t(c) - c.w$.

Case 2: The conflict involves independent sets from existing active conflict cores $\mathcal{C}' = \{c_1, \ldots, c_r\}$. By induction, we assume that each $c_j \in \mathcal{C}'$ satisfies that for any clique $C \subseteq V(c_j)$, $w(C) \leq w_t(c_j) - c_j.w$. Algorithm 2 merges these cores and the newly locked independent sets into a new core c, with $c.w = \beta + \sum_{j=1}^{r} c_j.w$.

We divide the sets in c into two groups. The first group consists of independent sets that do not belong to any core in $\mathcal{C}'$. If any independent set in this group does not contribute a vertex to a clique, the Core Invariant holds immediately.

Otherwise, all independent sets in the first group contribute vertices to the clique. The cores in $\mathcal{C}'$ are subsumed into c because, during propagation, the accumulated weight of empty independent sets $c_j.uw \geq c_j.w$, thereby satisfying the penalty of that core and allowing its remaining independent sets to re-enter conflict reasoning. Under this condition, the newly detected conflict necessarily forces at least one additional independent set, which is possibly from an unlocked core in $\mathcal{C}'$, to become empty, contributing a weight of at least β. This empty independent set is not used to satisfy the unlocking condition of its original core, but instead accounts for the penalty of the newly constructed conflict core.

Hence, the Core Invariant holds for the newly constructed core. By induction, it holds for all conflict cores maintained by Algorithm 2.

□

To illustrate how conflict cores participate in MaxSAT reasoning, we present an example abstracted from a real benchmark instance. The overall reasoning process is shown in Figure 2, and the analysis of the resulting penalty value of the core is illustrated in Figure 3.

Example 2. *Consider one independent set* $I_1 = \{v_1^{20}\}$ *and two existing conflict cores:*

$$c_1 = \{LI_1 = \{v_2^1, v_3^1\}, LI_2 = \{v_4^1, v_5^1\}, LI_3 = \{v_6^1\}\},$$

$$c_2 = \{LI_4 = \{v_7^5, v_8^5\}, LI_5 = \{v_9^5, v_{10}^5\}, LI_6 = \{v_{11}^5\}\},$$

where $c_1.w = 1$, $c_2.w = 5$, *and all locked independent sets in* c_1 *have weight 1 and those in* c_2 *have weight 5.*

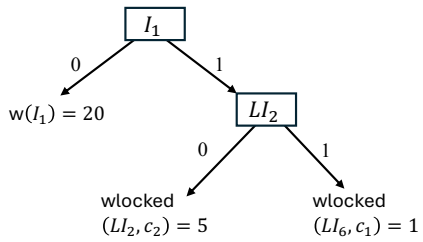

Figure 3: Analyze weight of new core

Algorithm 4: PrepareBES(Π, w_c)

Input: Independent sets Π, integer w_c
Output: Filtered independent sets Π'.

1 $w_{th} \leftarrow min(w_g, w_c/c_g)$, $\Pi' \leftarrow \emptyset$;
2 **for** $I_i \in \Pi$ **do**
3 $\quad$ **if** $w^*(I_i) \geq w_{th}$ **then** $\Pi' \leftarrow \Pi' \cup \{I_i\}$;
4 **return** Π';

Since I_1 is a unit independent set, the algorithm removes all non-neighbors of v_1 from all independent sets, namely v_2, v_3, v_4, v_7, and v_8. This makes LI_1 empty. As LI_1 is locked and $c_1.uw = w^*(LI_1) = c_1.w$, the Core Invariant of c_1 is satisfied and c_1 is unlocked, enabling LI_2 and LI_3 to participate in further reasoning. Similarly, LI_4 becomes empty, unlocking c_2 and allowing LI_5 and LI_6 to join the propagation. Next, $LI_2 = \{v_5\}$ becomes a unit unlocked independent set. Removing its non-neighbor v_{11} makes LI_6 empty. Since the core of LI_6 is unlocked, a new conflict is detected.

During conflict analysis, the algorithm starts from the empty set LI_6 and traces back all independent sets responsible for vertex removals until closure. Figure 3 provides an explanation of the resulting penalty of the conflict core. If I_1 contributes no vertex, the clique weight is reduced by 20. Otherwise, cores c_1 and c_2 are involved: since LI_1 and LI_4 cannot contribute vertices, either LI_2 also fails to contribute (yielding a reduction of 5), or LI_6 fails (yielding a reduction of 1). Hence, the minimum guaranteed reduction is selected.

After invoking AddCore, I_1 is split into $I'_1 = \{v_1^1\}$ and $I''_1 = \{v_1^{19}\}$, where I''_1 can participate in the reasoning process. Cores c_1 and c_2 are deactivated and merged into the new conflict core $c_3 = \{LI_1, LI_2, LI_3, LI_4, LI_5, LI_6, I'_1\}$, with total weight $c_3.w = c_1.w + c_2.w + min(20, 5, 1) = 7$. The weights of LI_1, LI_2, LI_3, and I'_1 are 1, while those of LI_4, LI_5, and LI_6 are 5.

3.3 Benefit-Enhanced Strategy

Example 1 shows that a single reasoning round reduces δ by only 1. When the gap $ub+\delta-t$ is large, this reduction is insufficient. Since the reduction in each round largely depends on the independent sets involved, it is crucial to focus on those that can yield a meaningful decrease in the bound.

Motivated by this observation, we propose a *Benefit-Enhanced Strategy* that adaptively preselects independent sets according to the current pruning demand and global graph characteristics. Algorithm 4 first computes a weight threshold $w_{th} = \min(w_g, w_c/c_g)$, where $w_c = \delta + ub - t$ is the remaining bound gap. Independent sets with weight below w_{th} are filtered out (line 1). The remaining sets with $w^*(I_i) \geq w_{th}$ are collected into Π' (lines 2–3) and passed to subsequent MaxSAT reasoning (line 4).

The parameter c_g is chosen based on the graph size n: $c_g = 2$ for $n \leq 5 \times 10^5$, $c_g = 3$ for $n \leq 10^6$, $c_g = 5$ for $n \leq 1.5 \times 10^6$, and $c_g = 8$ otherwise. The global threshold w_g is set to w_{avg}/c_g, where w_{avg} is the average vertex weight. For small graphs, branching is relatively cheap and aggressive pruning is unnecessary. For large graphs, branching becomes much more costly, and stronger pruning is therefore needed.

Algorithm 5: ULE-MaxSAT(G, O, B, Π, t, ub)

Input: $G = (V, E, w)$, vertex order O, branching set B, independent sets Π on $A = P \setminus B$, integer t, and upper bound $ub = w(G[A])$.
Output: The reduced branching set B.

1 $B = \{b_1, b_2, \cdots, b_{|B|}\}$, $b_1 < b_2 < \cdots < b_{|B|}$ *w.r.t.* O;
2 $\mathcal{L} \leftarrow \emptyset$ be the Ls, $\mathcal{C} \leftarrow \emptyset$ be the conflict cores;
3 **for** $i = |B|$ *down to 1* **do**
4 $\quad \Pi' \leftarrow \Pi$, $\mathcal{L}' \leftarrow \mathcal{L}$, $\mathcal{C}' \leftarrow \mathcal{C}$, $\delta \leftarrow w(b_i)$;
5 $\quad \Pi \leftarrow PrepareBES(\Pi, ub + \delta - t)$;
6 $\quad$ Let $I_1, \cdots, I_k$ with $I_j \cap \mathcal{N}(b_i) = \emptyset$ $(j = 1, \ldots, k)$;
7 $\quad$ **for** $j = 1$ *to* k **do**
8 $\quad\quad I_j \leftarrow I_j \cup \{b_i^{w^*(I_j)}\}$, $\delta \leftarrow \delta - w^*(I_j)$;
9 $\quad$ Let $U_1, \cdots, U_r$ with $|U_j \cap \mathcal{N}(b_i)| = 1$ $(j = 1, \ldots, r)$;
10 $\quad$ **for** $j = 1$ *to* r **do**
11 $\quad\quad$ Let $\mathcal{N}(b_i) \cap U_j = \{u\}$;
12 $\quad\quad$ **if** $\exists I_q \in \Pi$ s.t. $\mathcal{N}(u) \cap \mathcal{N}(b_i) \cap I_q = \emptyset$ **then**
13 $\quad\quad\quad F = \{I_q, U_j, \{b_i^\delta\}\}$;
14 $\quad\quad\quad (\Pi, \mathcal{C}, \mathcal{L}, \beta) = AddCore(F, \Pi, \mathcal{C}, \mathcal{L})$;
15 $\quad\quad\quad \delta \leftarrow \delta - \beta$;
16 $\quad\quad\quad$ **if** $ub + \delta \leq t$ **then break**;
17 $\quad$ **if** $ub + \delta > t$ **then**
18 $\quad\quad \Pi \leftarrow \Pi \cup \{\{b_i^\delta\}\}$;
19 $\quad\quad \delta \leftarrow$ *ULM-MaxSAT*$(\Pi, \mathcal{C}, \mathcal{L}, t, ub, \delta)$;
20 $\quad$ Restore I removed by *BES*;
21 $\quad$ **if** $ub + \delta \leq t$ **then**
22 $\quad\quad B \leftarrow B \setminus \{b_i\}$, $ub \leftarrow ub + \delta$;
23 $\quad$ **else**
24 $\quad\quad \Pi \leftarrow \Pi', \mathcal{L} \leftarrow \mathcal{L}', \mathcal{C} \leftarrow \mathcal{C}'$;
25 **return** B;

3.4 Enhanced MaxSAT Reasoning

Algorithm 5 presents the ULE-MaxSAT framework, which integrates ULM and BES to strengthen MaxSAT reasoning. It first initializes the branching set B, the locked independent sets $\mathcal{L}$, and the conflict cores $\mathcal{C}$ (lines 1–2). ULE-MaxSAT

Algorithm 6: GetBranches(G, t, O)

Input: $G = (V, E, w)$, an integer t and an ordering O
Output: The branching set B

$(B, \Pi, ub) \leftarrow$ get the initial independent sets;
if B *is not empty* **then**
  $B \leftarrow$ *ULE-MAXSAT*(G, O, B, Π, t, ub);
return B;

Algorithm 7: ULE-MWC(G)

Input: $G = (V, E, w)$
Output: Maximum weight clique C^*

1 $(C_0, O_0, G') \leftarrow$ *Initialize*$(G, 0)$;
2 $C^* \leftarrow C_0$, $V' \leftarrow$ vertices in G' ordered *w.r.t.* O_0;
3 **for** $i = |V'|$ *downto 1* **do**
4 $S \leftarrow \{v_i\}$, $P \leftarrow \mathcal{N}(v_i) \cap \{v_{i+1}, \cdots, v_{|V'|}\}$;
5 **if** $w(P) + w(S) > w(C^*)$ **then**
6 $(C'_0, O'_0, G'') \leftarrow$ *Initialize*$(G'[P], w(C^*) - w(S))$;
7 **if** $w(S) + w(C'_0) > w(C^*)$ **then**
8 $C^* \leftarrow S \cup C_0$;
9 $V'' \leftarrow$ vertices in G'' ordered *w.r.t.* O'_0
  $C_1 \leftarrow$ *BnB*(G'', V'', O'_0, S, C^*);
10 **if** $w(C_1) > w(C^*)$ **then** $C^* \leftarrow C_1$;
11 **return** C^*;

follows TSM-MWC [Jiang *et al.*, 2018] by prioritizing the detection of conflict cores involving two or three independent sets (lines 3-16), but instead of removing the conflicting parts, it maintains them in conflict cores for later reuse (line 14). After that, ULM-MaxSAT is applied to the remaining independent sets together with the stored cores, enabling further pruning of B (lines 17–19). The final branching set is then updated (lines 21–22). If no pruning can be applied, all independent sets, conflict cores, and locked independent sets are restored to their previous states (lines 23-24).

Algorithm 6 shows the *GetBranches* procedure used in the BnB framework. It first obtains the initial branching set B through partitioning independent sets Π, and then refines B using *ULE-MaxSAT*.

3.5 The Framework of ULE-MWC

Algorithm 7 presents the overall ULE-MWC framework. It follows the main structure of TSM-MWC and modifies only the MaxSAT reasoning within the BnB procedure. The algorithm initializes an initial clique, a vertex ordering, and a reduced graph, then explores candidate cliques using BnB, and finally returns the maximum weight clique found. Details of the preprocessing steps are omitted and can be found in WLMC and TSM-MWC.

4 Experimental Results

In this section, we compare ULE-MWC with two state-of-the-art BnB solvers, TSM-MWC and WLMC, on three benchmark datasets and present ablation studies to evaluate the effectiveness of ULM and BES.

4.1 Experiment Preliminaries

Benchmark Datasets. We evaluate ULE-MWC on three benchmark types:

- **DIMACS2 graphs**: 80 graphs up to 4000 vertices, densities 0.03–0.99[1].
- **Real-world massive graphs**: 77 graphs from the Network Data Repository [Rossi and Ahmed, 2015], including those used for WLMC and TSM-MWC[2].
- **Graphs from MWC practical applications**: Four application groups: 15 from error-correcting codes (ECC), 100 from kidney exchange schemes (KES), 129 from the Research Excellence Framework (REF), and 499 from the winner determination problem (WDP). The complete ECC, KES, and REF, along with a subset of the WDP instances, were collected by [McCreesh *et al.*, 2017][3].

Solvers. We evaluate our ULE-MWC[4] against two state-of-the-art BnB solvers TSM-MWC [Jiang *et al.*, 2018] and WLMC [Jiang *et al.*, 2017][5].

To evaluate the effectiveness of our ULM and BES, we generate two variants of ULE-MWC, called ULE-MWC$_{\text{noBES}}$ (removing BES) and ULE-MWC$_{\text{noULM}}$ (removing ULM).

Experimental Setup. ULE-MWC is implemented in C and tested on an AMD EPYC 7H12 server running Ubuntu 18.04. All experiments use a time limit of 10,000 seconds. We report the solution time (in seconds) and the number of explored branches ($\times 10^5$). A dash ("–") means that the solver fails to finish within the time limit. Best results are highlighted in bold.

4.2 Performance Comparison

TSM-MWC has proven to be very effective in practice, leaving little room for improvement. As a result, achieving better performance than it is a challenging task.

We first evaluate ULE-MWC on both DIMACS2 instances and real-world massive graphs (MASSIVE). As shown in Table 1, ULE-MWC consistently demonstrates superior or competitive performance compared with TSM-MWC and WLMC. On DIMACS2, ULE-MWC often achieves the smallest search tree or the fastest runtime, particularly on larger instances such as the brock800 and gen400 series, where it reduces solution time by several hundred seconds compared with TSM-MWC and significantly outperforms WLMC. Notably, on the Keller5 instance, neither TSM-MWC nor WLMC can obtain an exact solution within the time limit, whereas our ULE-MWC does, which highlights the strong pruning capability and high efficiency of our approach. ULE-MWC shows competitive or better results on

[1]available at http://archive.dimacs.rutgers.edu/Challenges/
[2]available at http://networkrepository.com
[3]available at https://doi.org/10.5281/zenodo.816293
[4]The source code of ULE-MWC is available at https://github.com/LlzmMing/ULE-MWC.git
[5]The source codes of both TSM-MWC and WLMC are available at https://home.mis.u-picardie.fr/˜cli/EnglishPage.html

Instance	$\|C\|$	ULE-MWC		TSM-MWC		WLMC	
		tree	time	tree	time	tree	time
DIMACS2							
brock400_1	3422	39.11	**80.12**	**38.55**	92.06	128.4	388.8
brock400_2	3350	50.61	**105.2**	**50.07**	120.0	162.4	483.7
brock400_3	3471	29.59	**62.48**	**29.43**	71.45	93.95	286.4
brock400_4	3626	50.99	**103.9**	**50.37**	120.5	150.5	441.7
brock800_1	3121	**715.1**	**1476**	726.89	1614	2309	5461
brock800_2	3043	**1010**	**1987**	1029	2190	3087	7093
brock800_3	3076	**840.6**	**1659**	854.2	1812	2647	6275
brock800_4	2971	**1055**	**2052**	1076	2263	3218	7267
C250.9	5092	2.072	**13.06**	**1.733**	16.79	14.23	111.3
DSJC1000_5	2186	**42.14**	**74.04**	42.96	81.39	138.8	202.2
gen400_p0.9_55	6718	476.1	**3985**	**340.6**	4729	-	-
gen400_p0.9_75	8006	23.81	**277.9**	**18.22**	315.5	147.1	2365
hamming10-2	50512	**0.007**	32.46	0.008	33.94	0.010	**2.549**
keller5	3317	**3015**	**8853**	-	-	-	-
MANN_a45	34265	6.637	**798.5**	8.618	1239	**5.307**	1277
p_hat1000-2	5777	1.975	**11.00**	**1.844**	12.86	10.06	75.23
p_hat1500-2	7360	53.84	**524.8**	**48.14**	551.1	469.2	5888
p_hat500-3	5375	1.661	**8.961**	**1.477**	10.91	10.73	91.51
p_hat700-3	7565	2.377	**19.89**	**2.131**	24.42	17.52	245.7
san400_0.9_1	9776	2.405	**48.56**	**2.044**	57.52	16.26	323.9
sanr400_0.7	2992	**10.27**	**18.48**	10.40	20.98	30.57	72.41
MASSIVE							
aff-digg	3836	**28.56**	**271.3**	28.84	315.8	164.1	735.5
aff-orkut-user2groups	971	**34.40**	**496.2**	34.40	507.9	74.27	626.2
bio-human-gene1	134713	25.99	6487	**25.99**	**6326**	-	-
bio-human-gene2	135310	**27.12**	**5099**	27.12	5459	28.15	6078
bio-mouse-gene	59952	46.47	1845	**46.30**	**1630**	55.51	4665
bn-...-864_...1-bg	32294	**6.050**	**1460**	6.599	1749	-	-
bn-...-864_...2-bg	27190	**5.689**	**1225**	5.751	1275	-	-
bn-...-865_...2-bg	29870	**2.713**	**1112**	2.725	1129	-	-
bn-...-867_...1-bg	29425	**3.867**	982.6	3.868	990.4	5.351	**723.7**
bn-...-867_...2-bg	36021	2.938	1112	**2.839**	1136	4.421	**805.9**
bn-...-868_...1-bg	31940	**7.510**	**1238**	7.591	1307	-	-
bn-...-868_...2-bg	29548	**7.414**	**1921**	8.257	2294	-	-
bn-...-869_...1-bg	27957	**3.314**	**1170**	3.316	1186	6.168	1221
bn-...-869_...2-bg	29250	**4.496**	**1829**	4.519	1874	15.73	4680
bn-...-870_...1-bg	28810	**7.050**	**1101**	7.067	1124	19.25	5924
bn-...-870_...2-bg	35415	**6.055**	**1516**	6.100	1544	-	-
bn-...-871_...1-bg	37828	**13.31**	1456	13.32	1475	16.42	**1202**
bn-...-871_...2-bg	32835	**8.119**	**1717**	8.454	1880	-	-
bn-...-872_...2-bg	35698	**13.44**	**1455**	13.84	1760	-	-
bn-...-873_...1-bg	29944	11.10	5210	**7.501**	**4553**	-	-
bn-...-873_...2-bg	32445	**3.593**	954.3	3.606	969.1	5.979	**921.6**
bn-...-874_...2-bg	30885	**6.454**	1490	6.476	1510	10.27	**1453**
bn-...-876_...1-bg	50355	**27.73**	1398	27.76	**1392**	-	-
bn-...-876_...2-bg	33085	**6.310**	**1599**	6.373	1662	-	-
bn-...-878_...1-bg	27775	**3.108**	**830.3**	3.121	837.9	11.23	2228
bn-...-886_...1	26281	6.343	**1332**	**5.444**	1380	-	-
bn-...-889_...1	27500	**9.518**	**2288**	9.678	2867	-	-
bn-...-889_...2	24771	**4.822**	914.7	4.833	926.9	6.614	**735.6**
bn-...-912_...2	35063	**3.662**	**965.9**	3.686	972.9	9.248	2650
rec-epinions	1054	**3.448**	**45.58**	3.448	57.67	7.685	45.8
soc-livejournal-user-groups	1054	**22.33**	**127.9**	22.33	134.7	31.69	134.7
soc-orkut	5452	**5.635**	126.1	5.639	**120.1**	23.06	135.1
soc-orkut-dir	6147	**4.512**	**141.9**	4.516	158.4	27.23	145.1
tech-ip	668	**1.369**	134.4	**1.369**	184.0	2.674	**131.5**
tech-p2p	18897	**5.542**	**712.2**	8.294	997.2	-	-
twitter_mpi	13524	**3.359**	**640.6**	4.295	843.0	33.51	1916
web-wikipedia_link_it	89947	**2.034**	187.5	**2.334**	196.9	2.122	**142.1**

Table 1: Comparison of ULE-MWC with TSM-MWC and WLMC on DIMACS2 and MASSIVE graphs, using a 10,000s cutoff. Best results are in bold. Only representative instances are reported: difficult instances unsolved by all solvers, trivial instances solved within 10s by all, and cases with time differences below 5s are omitted.

massive graphs. It achieves the fastest runtime on most brain-network instances, highlighting its efficiency in handling large-scale graphs.

We further test ULE-MWC on practical MWCP applications, including ECC, KES, REF, and WDP instances, and summarize the results in Table 2. ULE-MWC solves all 15 ECC and 499 WDP instances, and 84 KES and 106 REF instances, matching or exceeding the number of instances solved by TSM-MWC and WLMC. In terms of runtime, ULE-MWC consistently outperforms the baselines, achieving the lowest average solution time across all four datasets.

Dataset	ULE-MWC		TSM-MWC		WLMC	
	#	avgt	#	avgt	#	avgt
ECC(15)	**15**	**10.10**	**15**	10.42	**15**	10.44
KES(100)	**84**	**150.8**	**84**	163.1	82	202.7
REF(129)	**106**	**1.950**	**106**	1.988	**106**	2.375
WDP(499)	**499**	**3.353**	**499**	4.197	**499**	12.71

Table 2: Number of solved graphs (#) and average runtime (s) for ULE-MWC, TSM-MWC, and WLMC on MWCP practical applications. Total graphs per group are in brackets in the first column.

	ULE-MWC			ULE-MWC$_{\text{noBES}}$			ULE-MWC$_{\text{noULM}}$		
	#	tree	avgt	#	tree	avgt	#	tree	avgt
DIMACS(80)	**68**	109.4	**326.8**	67	**103.7**	405.1	**68**	117.7	361.7
MASSIVE(77)	**77**	**7.625**	**720.8**	**77**	7.637	792.6	**77**	7.690	803.0
ECC(15)	**15**	**12.83**	**10.11**	**15**	12.85	11.08	**15**	14.24	10.17
KES(100)	**84**	179.4	**150.8**	**84**	**145.4**	152.1	**84**	145.6	152.0
REF(129)	**106**	**3.444**	1.950	**106**	3.470	2.415	**106**	3.789	**1.876**
WDP(499)	**499**	2.113	**3.353**	**499**	**1.894**	4.315	**499**	2.065	3.404

Table 3: The number of solved graphs (#), the mean search tree size in 10^5 (tree), and the average solution time (avgt) of ULE-MWC, ULE-MWC$_{\text{noBES}}$ and ULE-MWC$_{\text{noULM}}$.

For the KES instances, we further compute the average runtime over the eight instances requiring more than 5 seconds: TSM-MWC needs 1710s on average, while ULE-MWC only requires 1580s, indicating better efficiency.

4.3 Ablation Studies

To evaluate the effectiveness of our proposed strategies, we conduct ablation studies on ULE-MWC by removing the BES (ULE-MWC$_{\text{noBES}}$) and ULM (ULE-MWC$_{\text{noULM}}$) components. As shown in Table 3, removing BES generally increases runtime across most datasets, especially on DIMACS and MASSIVE instances, indicating that the BES strategy effectively accelerates the search process. Similarly, ULE-MWC$_{\text{noULM}}$ exhibits larger search trees on DIMACS and MASSIVE datasets compared with the full ULE-MWC, confirming that the ULM strategy contributes to efficient pruning of the search space.

5 Conclusion

In this paper, we analyze the limitations of existing MaxSAT-based bounding for the Maximum Weight Clique Problem. Independent sets in conflicts are discarded after one use, and conflicts are processed in a fixed order regardless of their pruning effectiveness, which restricts the reuse of conflict information and weakens bound tightening. To address these issues, we introduce two complementary techniques. The UnLocking Mechanism preserves conflict cores to enable the reuse of informative independent sets, while the Benefit-Enhanced Strategy guides MaxSAT reasoning toward conflicts with higher pruning potential. By integrating them into a unified BnB solver, ULE-MWC, we obtain consistently tighter upper bounds and superior performance over state-of-the-art MWCP solvers on standard benchmarks.

Acknowledgements

This work is partially supported by the French National Research Agency (ANR) under the Chair MASSAL'IA (ANR-19-CHIA-0013-01), co-funded by the French electricity distribution network operator Enedis.

References

[Butenko and Wilhelm, 2006] Sergiy Butenko and Wilbert E Wilhelm. Clique-detection models in computational biochemistry and genomics. *European Journal of Operational Research*, 173(1):1–17, 2006.

[Cai and Lin, 2016] Shaowei Cai and Jinkun Lin. Fast solving maximum weight clique problem in massive graphs. In *IJCAI*, pages 568–574, 2016.

[Fan *et al.*, 2017] Yi Fan, Nan Li, Chengqian Li, Zongjie Ma, Longin Jan Latecki, and Kaile Su. Restart and random walk in local search for maximum vertex weight cliques with evaluations in clustering aggregation. In *IJCAI-17*. International Joint Conferences on Artificial Intelligence (IJCAI), 2017.

[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. *Journal of Artificial Intelligence Research*, 55:799–833, 2016.

[Hartmanis, 1982] Juris Hartmanis. *Computers and intractability: a guide to the theory of np-completeness (michael r. garey and david s. johnson)*, volume 24. Society for Industrial and Applied Mathematics, 1982.

[Jiang *et al.*, 2017] Hua Jiang, Chu-Min Li, and Felip Manya. An exact algorithm for the maximum weight clique problem in large graphs. In *Proceedings of the AAAI conference on artificial intelligence*, volume 31, 2017.

[Jiang *et al.*, 2018] Hua Jiang, Chu-Min Li, Yanli Liu, and Felip Manya. A two-stage maxsat reasoning approach for the maximum weight clique problem. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[Li and Quan, 2010] Chu-Min Li and Zhe Quan. An efficient branch-and-bound algorithm based on maxsat for the maximum clique problem. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 24, pages 128–133, 2010.

[Li *et al.*, 2017] Chu-Min Li, Hua Jiang, and Felip Manyà. On minimization of the number of branches in branch-and-bound algorithms for the maximum clique problem. *Computers & Operations Research*, 84:1–15, 2017.

[Li *et al.*, 2018] Chu-Min Li, Zhiwen Fang, Hua Jiang, and Ke Xu. Incremental upper bound for the maximum clique problem. *INFORMS Journal on Computing*, 30(1):137–153, 2018.

[Mascia *et al.*, 2010] Franco Mascia, Elisa Cilia, Mauro Brunato, and Andrea Passerini. Predicting structural and functional sites in proteins by searching for maximum-weight cliques. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 24, pages 1274–1279, 2010.

[McCreesh *et al.*, 2017] Ciaran McCreesh, Patrick Prosser, Kyle Simpson, and James Trimble. On maximum weight clique algorithms, and how they are evaluated. In *International conference on principles and practice of constraint programming*, pages 206–225. Springer, 2017.

[Östergård, 2002] Patric RJ Östergård. A fast algorithm for the maximum clique problem. *Discrete Applied Mathematics*, 120(1-3):197–207, 2002.

[Rossi and Ahmed, 2015] Ryan Rossi and Nesreen Ahmed. The network data repository with interactive graph analytics and visualization. In *Proceedings of the AAAI conference on artificial intelligence*, volume 29, 2015.

[San Segundo *et al.*, 2019] Pablo San Segundo, Fabio Furini, and Jorge Artieda. A new branch-and-bound algorithm for the maximum weighted clique problem. *Computers & Operations Research*, 110:18–33, 2019.

[Sanroma *et al.*, 2016] Gerard Sanroma, Adrian Penate-Sanchez, René Alquézar, Francesc Serratosa, Francesc Moreno-Noguer, Juan Andrade-Cetto, and Miguel Ángel González Ballester. Msclique: multiple structure discovery through the maximum weighted clique problem. *Plos one*, 11(1):e0145846, 2016.

[Wang *et al.*, 2020] Yiyuan Wang, Shaowei Cai, Jiejiang Chen, and Minghao Yin. Sccwalk: An efficient local search algorithm and its improvements for maximum weight clique problem. *Artificial Intelligence*, 280:103230, 2020.

[Wu *et al.*, 2012] Qinghua Wu, Jin-Kao Hao, and Fred Glover. Multi-neighborhood tabu search for the maximum weight clique problem. *Annals of Operations Research*, 196(1):611–634, 2012.

[Zhang *et al.*, 2014] Dong Zhang, Omar Javed, and Mubarak Shah. Video object co-segmentation by regulated maximum weight cliques. In *Computer Vision - ECCV 2014 - 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part VII*, volume 8695 of *Lecture Notes in Computer Science*, pages 551–566. Springer, 2014.

[Zhian *et al.*, 2013] Hootan Zhian, Masoud Sabaei, Nastooh Taheri Javan, and Omid Tavallaie. Increasing coding opportunities using maximum-weight clique. In *2013 5th Computer Science and Electronic Engineering Conference (CEEC)*, pages 168–173. IEEE, 2013.

[Zhou *et al.*, 2017] Yi Zhou, Jin-Kao Hao, and Adrien Goëffon. Push: A generalized operator for the maximum vertex weight clique problem. *European Journal of Operational Research*, 257(1):41–54, 2017.