A Fast Exact Solver with Theoretical Analysis for the Maximum Edge-Weighted Clique Problem

Lu Liu, Mingyu Xiao*, Yi Zhou

University of Electronic Science and Technology of China
2w6f8c@gmail.com, myxiao@uestc.edu.cn, zhou.yi@uestc.edu.cn

Abstract

The maximum vertex-weighted clique problem (MVWCP) and the maximum edge-weighted clique problem (MEWCP) are two natural extensions of the fundamental maximum clique problem. In this paper, we systematically study MEWCP and make the following major contributions: (1) We show that MEWCP is NP-hard even when the minimum degree of the graph is $n-2$, in contrast to MVWCP which is polynomial-time solvable when the minimum degree of the graph is at least $n-3$. This result distinguishes the complexity of the two problems for the first time. (2) To address MEWCP, we develop an efficient branch-and-bound algorithm called MEWCat with both practical and theoretical performance guarantees. In practice, MEWCat utilizes a new upper bound tighter than existing ones, which allows for more efficient pruning of branches. In theory, we prove a running-time bound of $O^*(1.4423^n)$ for MEWCat, which breaks the trivial bound of $O^*(2^n)$ in the research line of practical exact MEWCP solvers for the first time. (3) Empirically, we evaluate the performance of MEWCat on various benchmark instances. The experiments demonstrate that MEWCat outperforms state-of-the-art exact solvers significantly. For instance, on 16 DIMACS graphs that the state-of-the-art solver BBEWC fails to solve within 7200 seconds, MEWCat solves all of them with an average time of less than 1000 seconds. On real-world graphs, MEWCat achieves an average speedup of over 36x.

Introduction

A *clique* is a graph such that every pair of vertices in it are adjacent. The *maximum clique problem* (MCP), to find a clique of maximum cardinality from a given graph, is a classical NP-hard problem that has been extensively studied in optimization (Gary and Johnson 1979; Karp 2010).

There are several variants of MCP. The *maximum vertex-weighted clique problem* (MVWCP) is to find a clique in a vertex-weighted graph such that the total vertex weight is maximized. The *maximum edge-weighted clique problem* (MEWCP) is to find a clique in an edge-weighted graph such that the total edge weight is maximized. In the maximum total-weighted clique problem (MTWCP), both vertices and edges are weighted and the goal is to maximize the total vertex and edge weight in the clique. Please see (Wu and Hao 2015) for an introduction to MCP and its variants. All these variants are NP-hard since MCP is a special case of them.

To find a clique in a graph G is equal to finding an independent set in the complement graph of G. Therefore, we can get the corresponding variants of the maximum independent set problem. The maximum clique problem and the corresponding maximum independent set problem may have the same computational complexity. However, in terms of practical solvers, the algorithms for the pair of problems may be different since the complement of a sparse graph will become dense, and usually fast practical algorithms on sparse and dense graphs are different.

In this paper, we focus on MEWCP. Because of edge weights, MEWCP is more flexible than MCP in modeling problems arising in different fields, such as material science (Agapito et al. 2016), bioinformatics (Butenko and Wilhelm 2006; Li et al. 2010; Tomita, Akutsu, and Matsunaga 2011), computer vision and pattern recognition (Ma and Latecki 2012; San Segundo and Artieda 2015), robotics (San Segundo and Rodriguez-Losada 2013), and so on. Though important, MEWCP is less studied compared to MVWCP, and most existing algorithms for MVWCP cannot be directly applied to MEWCP. This is unusual and we believe that there is something essentially different between the two variants. All above motivated us to carefully study the complexity of MEWCP.

In terms of algorithms, there are several heuristic and exact algorithms to address MEWCP. Pullan (2008) proposed a local search algorithm that switches three phases to avoid local optimum. An evolutionary algorithm is proposed by Fontes, Goncalves, and Fontes (2018). Li et al. (2018b) presented a local search algorithm based on multiple rules to select the added vertex or the swapped vertex pair. A stochastic local search algorithm, which combines clique construction, local search, and graph reduction, is presented in (Chu et al. 2020). The heuristic algorithms can obtain solutions of high quality in a short time, but cannot guarantee the optimality of the solutions obtained.

Existing exact algorithms for MEWCP can be divided into two approaches. One approach is to formulate mathematical programming and then use existing solvers to solve the model (Gouveia and Martins 2015; Shimizu, Yamaguchi,

*Corresponding author

and Masuda 2017). The other approach is the branch-and-bound method. Hosseinian, Fontes, and Butenko (2018) proposed a quadratic programming formulation for MEWCP and used its continuous relaxation as an upper bound to prune the search. Shimizu, Yamaguchi, and Masuda (2019) presented EWCLIQUE, an algorithm that iteratively computes optimal solutions to subproblems of increasing size. San Segundo et al. (2019) introduced a new upper bound for MEWCP, which shifts edge weights to vertices, and obtained the algorithm BBEWC. Shimizu, Yamaguchi, and Masuda (2020) proposed a new branch-and-bound algorithm called MECQ, which assigns edge weights to vertices and computes upper bounds using vertex coloring. Hosseinian, Fontes, and Butenko (2020) proposed a new analytic upper bound on the clique number of a graph, and embedded it in a branch-and-bound framework. They mainly focused on the theory of the new upper bound, and the performance of their algorithm obtained is worse than both MECQ and BBEWC according to the results reported.

In summary, this paper makes the following contributions.

- We distinguish the computational complexity of MVWCP and MEWCP for the first time. We consider graphs of bounded degree. It is less interesting to consider graphs with bounded maximum degree, since the size of a maximum clique will also be bounded by the maximum degree. Therefore, we consider graphs with bounded minimum degree. We prove that, surprisingly, MEWCP is NP-hard even when the minimum degree of the graph is $n-2$, in contrast to MVWCP which can be solved in polynomial time when the minimum degree of the graph is at least $n-3$. See Table 1 for the computational complexity of MVWCP and MEWCP.
- We develop an efficient branch-and-bound solver called MEWCat for MEWCP. Our algorithm utilizes a new upper bound tighter than existing ones. This new bound allows for more efficient pruning of search nodes, resulting in a faster algorithm. Experiments on DIMACS benchmarks, real-world graphs, and random graphs show that MEWCat outperforms existing solvers significantly. For example, on 16 DIMACS instances that the state-of-the-art solver BBEWC fails to solve within 7200 seconds, MEWCat solves all of them with an average time of less than 1000 seconds. On real-world graphs, MEWCat achieves an average speedup of over 36x.
- We theoretically prove a running-time bound of $O^*(1.4423^n)$[1] for MEWCat. Most previous algorithms for MEWCP do not analyze the running-time bound or have a bound of $O^*(2^n)$. This is the first time we break the trivial bound of $O^*(2^n)$ in the research line of practical exact solvers for MEWCP.

Preliminaries

Given an edge-weighted undirected graph $G=(V,E,w)$, we use $w(u,v)$ to denote the weight of an edge $(u,v)\in E$. Let $\delta(G)$ denote the minimum degree of graph G. Denote the set of the neighbors of a vertex v by $N(v)=\{u\in V \mid (u,v)\in E\}$. Let $G[S]$ be the subgraph of G induced by $S\subseteq V$, and let $E(S)$ be the edge set of $G[S]$. A set $I\subseteq V$ is an *independent set* if every two vertices in I are not adjacent. An *independent set partition* of V is a partition of V such that each part is an independent set. A set $C\subseteq V$ is a *clique* if every two vertices in C are adjacent. A clique is *maximal* if no larger clique contains it. For a clique C, we use $W(C)$ to denote the total edge weight of C, i.e., $W(C)=\sum_{(u,v)\in E(C)} w(u,v)$. For a clique C and a vertex $v\in V$, let $W(C,v)=\sum_{u\in C} w(u,v)$.

[1]The O^* notation ignores the polynomial factor.

Problems	P	NP-hard
MVWCP	$\delta(G)\geq n-3$ (Xiao et al. 2021)	$\delta(G)\leq n-4$ (Garey et al. 1974)
MEWCP	$\delta(G)=n-1$ (Trivial)	$\delta(G)\leq n-2$ (Theorem 1 in this paper)

Table 1: The computational complexity of MVWCP and MEWCP on graphs with different minimum degree (denoted by $\delta(G)$).

Computational Complexity

First of all, we show some differences between the vertex-weighted and edge-weighted versions of the maximum clique problem by carefully checking the computational complexity. As mentioned before, both MVWCP and MEWCP are NP-hard on general graphs even when the weights are identical. Now we consider the complexity of the two problems on graphs close to complete graphs (i.e., each vertex is not adjacent to only a few vertices). When the minimum degree $\delta(G)$ of the graph is $n-1$, the whole graph is a clique and the two problems are trivial. When $\delta(G)$ is at least $n-3$, MVWCP is equivalent to the maximum independent set problem in vertex-weighted graphs with maximum degree 2. The latter problem can be solved in polynomial time by using the reduction rules in (Xiao et al. 2021) to iteratively handle degree-1 and degree-2 vertices. Thus, MVWCP can be solved in polynomial time when $\delta(G)\geq n-3$. On the other hand, MVWCP becomes NP-hard when $\delta(G)\leq n-4$ since the maximum independent set problem is NP-hard in degree-3 graphs (Garey, Johnson, and Stockmeyer 1974). For MEWCP, the problem seems harder. We will show that MVWCP becomes NP-hard even when $\delta(G)=n-2$ (i.e., each vertex in the graph is not adjacent to at most one other vertex). The computational complexity of MVWCP and MEWCP is summarized in Table 1.

Theorem 1. *MEWCP is NP-hard on graphs where each vertex is not adjacent to at most one other vertex.*

Proof. We prove the theorem by reducing from MAX-2-SAT. MAX-2-SAT is NP-hard (Garey, Johnson, and Stockmeyer 1974), and it is to find an assignment to satisfy the maximum number of clauses simultaneously in a conjunctive normal form (CNF) with each clause containing exactly two literals.

Let X be a MAX-2-SAT instance contains n variables $x_1,\ldots,x_n$ and m different clauses $c_1,\ldots,c_m$, allowing for clause duplication. Assume that each clause c_i appears w_i

times for $i \in \{1, \dots, m\}$ and we let $W = 1 + \max\{w_i \mid i \in \{1, \dots, m\}\}$. We construct an instance G of MEWCP as follows. The graph G contains $n' = 2n$ vertices: for each variable x_i in X, we introduce two vertices x_i and $\neg x_i$. For each $i \in \{1, \dots, n\}$, there is an edge between vertex x_i and each other vertex except $\neg x_i$. Thus, $\delta(G) = n' - 2$. Next, we set the edge weight of G. Initially, each edge has a weight of W. Then, for each clause $c_i = x \vee y$, we change the edge weight between $\neg x$ and $\neg y$ to $W - w_i$.

We show that there are k clauses in X (counting duplications) that can be satisfied simultaneously if and only if G has a maximal clique with edge-weight $\frac{n(n-1)}{2}W - \sum_{i=1}^{m} w_i + k$.

Consider a solution to X that satisfies k clauses. For each variable x_i, if x_i is set to $true$, include vertex x_i to the clique, otherwise include $\neg x_i$. We will get a clique H of size n, which is maximal. We show that the edge weight of H is $\frac{n(n-1)}{2}W - \sum_{i=1}^{m} w_i + k$. If all edges have the same weight W, the weight of the clique would be $\frac{n(n-1)}{2}W$. If a clause $c_i = x \vee y$ is not satisfied, the edge $(\neg x, \neg y)$ will be in the clique H. Note that the weight of this edge is $W - w_i$ and then the weight of the clique will decrease by w_i. On the other hand, if a clause $c_i = x \vee y$ is satisfied, the edge $(\neg x, \neg y)$ is not in the clique H. There are $\sum_{i=1}^{m} w_i$ clauses in total. The solution to X satisfies k clauses and then there are $\sum_{i=1}^{m} w_i - k$ clauses not satisfied. Thus, the edge weight of the clique H will decrease $\sum_{i=1}^{m} w_i - k$ from $\frac{n(n-1)}{2}W$ in total. So the edge weight of H is $\frac{n(n-1)}{2}W - \sum_{i=1}^{m} w_i + k$.

For the reverse direction, let H be a maximal clique in G with weight $W(H) = \frac{n(n-1)}{2}W - \sum_{i=1}^{m} w_i + k$. since H is maximal, exactly one of x_i and $\neg x_i$ is in H for $i \in \{1, \dots, n\}$. By setting the one included to $true$, we get an assignment A for X. Consider a clause $c_i = x \vee y$, the edge $(\neg x, \neg y)$ is in H if and only if both x and y are $false$. Let C' be the set of clauses $c_i = x \vee y$ such that edge $(\neg x, \neg y)$ is in H. We have $W(H) = \frac{n(n-1)}{2}W - \sum_{c_i \in C'} w_i$. Thus, $\sum_{c_i \in C'} w_i = \frac{n(n-1)}{2}W - W(H) = \sum_{i=1}^{m} w_i - k$, which is the number of clauses not satisfied by A. Thus, the number of clauses satisfied by A is k.

Since the maximum edge-weighted clique is a maximal clique, at most k clauses can be simultaneously satisfied in X is equivalent to that the maximum edge-weighted clique in G has weight $\frac{n(n-1)}{2}W - \sum_{i=1}^{m} w_i + k$. $\square$

The Branch-and-Bound Framework

Although there are many existing algorithms for MVWCP, they can not be directly modified for MEWCP. The above computational complexity can also explain that MEWCP may be harder than MVWCP. In this paper, we focus on branch-and-bound algorithms for MEWCP.

The simple and basic idea of a branch-and-bound algorithm is to branch on a vertex by either deleting it from the graph or including it in the solution. This operation will result in the problem to find a maximum clique containing a set C of some pre-decided vertices. Note that C should also form a clique, otherwise there is no solution. To get an effective algorithm, we may reduce the search space by setting more constraints. First, we may determine a vertex set S such that the maximum clique only contains vertices in $C \cup S$. Thus, we only need to consider vertices in S instead of the whole vertex set $V \setminus C$. Second, we may already get a current clique C_{max}, which can be obtained by some heuristic algorithms or previous steps of the branch-and-bound algorithm. Then, we can prune the branches that cannot lead to a clique better than the current clique C_{max}.

```
Algorithm 1: MEWCat($G$)
1: $C_{max} \leftarrow PLS(G)$
2: return $Search(G, \emptyset, V, C_{max})$
```

```
Algorithm 2: Search($G$, $C$, $S$, $C_{max}$)
1: if $W(C) > C_{max}$ then
2:     $C_{max} \leftarrow C$
3: end if
4: $B \leftarrow GetBranches(G, C, S, C_{max})$
5: $B' \leftarrow GetBranchesDegree(G, S)$
6: if $|B| > |B'|$ then
7:     $B \leftarrow B'$
8: end if
9: for all $v \in B$ do
10:    $C_{max} \leftarrow Search(G, C \cup \{v\}, S \cap N(v), C_{max})$
11:    $S \leftarrow S \setminus \{v\}$
12: end for
13: return $C_{max}$
```

Based on the above ideas, we propose a new branch-and-bound algorithm MEWCat to address MEWCP. The entry of MEWCat is Algorithm 1. The input is an edge-weighted graph $G = (V, E, w)$. It first computes an initial solution C_{max} with the heuristic algorithm PLS (Pullan 2008), and then calls *Search* to solve the problem. MECQ (Shimizu, Yamaguchi, and Masuda 2020) also uses PLS to compute the initial solution, and our parameter settings of PLS are the same as those in it.

The function *Search* is responsible for searching a clique F (if exists) in G such that the weight is maximized under the constraints that $C \subseteq F \subseteq C \cup S$ and $W(F) > W(C_{max})$. The pseudocode is shown in Algorithm 2.

The main idea of *Search* is to compute a branching set $B \subseteq S$ such that, if the maximum edge-weighted clique F under the constraints that $C \subseteq F \subseteq C \cup S$ and $W(F) > W(C_{max})$ exists, then it must contain at least one vertex in B. Then, we simply branch by including each vertex in B to C to generate a new search node. We will use two methods *GetBranchesDegree* and *GetBranches* to compute B and adopt the smaller one returned by them.

To get a more efficient algorithm, we also use some tricks when branching on B. First, the vertices in B are processed in ascending order of the degree in $G[S]$. Second, to avoid duplicated subproblems, when a vertex $v \in B$ is processed in a branch, we delete v from S in the remaining branches (Step 11). These two techniques are helpful. For the vertices v processed first, since the degree is small, the set $N(v) \cap S$

is also small; and for the vertices v processed last, the set $N(v) \cap S$ may also be small since S will become small after removing the processed vertices from it. Accordingly, the generated subproblems are balanced to some degree, which could potentially reduce the size of the search tree in the algorithm.

The way to compute the set B is important. Next, we consider *GetBranchesDegree* and *GetBranches*. The first method *GetBranchesDegree* is relatively simpler. The second method *GetBranches* needs to use a new upper bound and the details will be introduced later.

- *GetBranchesDegree*: It finds a vertex v with the maximum degree in the induced graph $G[S]$ and lets B be the set of vertices in S not adjacent to v and vertex v itself. Note that if the clique does not contain any vertex not adjacent to v, then we can include v to the clique (if v is not in the clique) to get a larger clique. Thus, at least one vertex in B will be in the maximum clique.
- *GetBranches*: The idea is to compute a pruned set P such that expanding C with vertices only from P will not lead to a clique better than the current clique C_{max}. Then, we return $B = S \setminus P$. This idea is common in the literature (Li, Jiang, and Manyà 2017; Li et al. 2018a). We show how to compute P later.

The purpose of *GetBranches* is also to compute a small set B. Meanwhile, it cannot be too time-consuming; otherwise, the total time may be long, even if it finds a small set. Therefore, its pruning capability and running time should be well balanced. In the sequel, we focus on developing an efficient *GetBranches* function based on a new upper bound for MEWCP.

A New Upper Bound

Consider the problem of finding the maximum edge-weighted clique F under the constraint $C \subseteq F \subseteq C \cup S$. we assume that C already forms a clique, and each vertex in S is adjacent to all vertices in C. We first reduce this problem to the maximum vertex-and-edge-weighted clique problem (MTWCP).

Given G, C, and S, we construct a vertex-and-edge-weighted graph $G' = (V', E')$ with $V' = S$ and $E' = E(S)$. The edge weight in G' is the same as that in the original graph G. For every vertex $v \in V'$, the vertex weight $w(v)$ is defined to be $W(C, v)$.

For a clique F in G with $C \subseteq F \subseteq C \cup S$, we let $F' = F \setminus C$. Note that F' is a clique in S. We have $W(F) = W(C \cup F') = W(C) + W(F') + \sum_{v \in F'} W(C, v)$. Note that $W(F') + \sum_{v \in F'} W(C, v)$ is the sum of edge weight and vertex weight of the clique F' in G'. For a fixed C, the part $W(C)$ is also fixed, and then the problem becomes to find a maximum vertex-and-edge-weighted clique F' in G'. To upper bound $W(F)$ in G, it suffices to upper bound the maximum vertex-and-edge-weighted clique in G'.

A frequently used technique to compute an upper bound for the maximum clique problems is the *independent set partition* technique (Tomita and Seki 2003). We also use this technique here.

We consider the vertex-and-edge-weighted graph $G' = (V', E')$ and want to find a maximum clique F' in it. Let $I_1, I_2, \ldots, I_k$ be an independent set partition of $V' = S$. For each $v \in S$, let $\tau(v)$ be the index of the independent set that v belongs to, i.e., $v \in I_{\tau(v)}$. For an edge between v and u, previously we use $w(v, u) = w(u, v)$ to denote the weight of the edge since the edge has no direction. Now, we slightly modify the weight function w to distinguish $w(v, u)$ and $w(u, v)$. For an edge (v, u), define weight function w' as follows: if $\tau(v) > \tau(u)$ then $w'(v, u) = w(v, u)$; otherwise $w'(v, u) = 0$ for $\tau(v) < \tau(u)$. Thus, $w(v, u) = w(u, v) = w'(v, u) + w'(u, v)$. For a clique F' in G' and a vertex $v \in F'$, we define the total weight *covered* by v in F' by

$$cover(F', v) = w(v) + \sum_{u \in F'} w'(v, u).$$

Observe that two vertices in the same independent set cannot be in the same clique. Thus, $|F' \cap I_i| \le 1$ holds for each $i \in \{1, \ldots, k\}$. Now we analyze an upper bound for $cover(F', v)$ based on this observation. It is not hard to see that $cover(F', v) = w(v) + \sum_{u \in F'} w'(v, u) = w(v) + \sum_{i=1}^{k} \sum_{u \in I_i \cap F'} w'(v, u) = w(v) + \sum_{i<\tau(v)} \sum_{u \in I_i \cap F'} w(v, u) \le w(v) + \sum_{i<\tau(v)} \max\{w(v, u) \mid u \in I_i \cap F'\} \le w(v) + \sum_{i<\tau(v)} \max\{w(v, u) \mid u \in I_i \cap N(v)\}$.

Thus, $cover(F', v)$ is upper bounded by

$$\sigma(v) := W(C, v) + \sum_{i<\tau(v)} \max\{w(v, u) \mid u \in I_i \cap N(v)\}.$$

The total weight of a clique F' in G' is equal to $\sum_{v \in F'} w(v) + \sum_{(v,u) \in E(F')} w(v, u) = \sum_{v \in F'} w(v) + \sum_{v \in F'} \sum_{u \in F'} w'(v, u) = \sum_{v \in F'} cover(F', v) \le \sum_{v \in F'} \sigma(v) \le \sum_{i=1}^{k} \max\{\sigma(v) \mid v \in I_i\}$. So, for every clique F in G with $C \subseteq F \subseteq C \cup S$, we get the new bound

$$W(F) \le W(C) + \sum_{i=1}^{k} \max\{\sigma(v) \mid v \in I_i\}. \qquad (1)$$

This bound is equivalent to the one proposed by Shimizu, Yamaguchi, and Masuda (2020).

Next, we derive a tighter bound. We sort the vertices in S in ascending order of $\tau(\cdot)$ (for vertices with the same $\tau(\cdot)$, we order them arbitrarily). For a vertex $v_j \in S$, we consider cliques in which v_j is the maximum vertex in this order, i.e., we are considering cliques F with $C \cup \{v_j\} \subseteq F \subseteq C \cup \{v_1, \ldots, v_j\}$. For such cliques, we denote the upper bound of their weight by $upper(v_j)$.

For a clique F with $C \subseteq F \subseteq C \cup S$, let v_j be the maximum vertex in it, then $W(F) \le upper(v_j)$. Therefore, we get the bound

$$W(F) \le \max\{upper(v) \mid v \in S\}. \qquad (2)$$

We can compute $upper(v_j)$ as follows. For a clique F with $C \cup \{v_j\} \subseteq F \subseteq C \cup \{v_1, \ldots, v_j\}$, we have $C \cup \{v_j\} \subseteq F \subseteq (C \cup \{v_j\}) \cup (\{v_1, \ldots, v_{j-1}\} \cap N(v_j))$. So the candidate set S' for F is $\{v_1, \ldots, v_{j-1}\} \cap$

$N(v_j) = \bigcup_{i<\tau(v_j)} I_i \cap N(v_j)$. Let $C' \leftarrow C \cup \{v_j\}$ and $I'_i \leftarrow I_i \cap N(v_j)$ for $i < \tau(v_j)$, then the new $\sigma(\cdot)$ for $v \in S'$ becomes $\sigma'(v) = W(C \cup \{v_j\}, v) + \sum_{i<\tau(v)} \max\{w(u,v) \mid u \in I_i \cap N(v_j) \cap N(v)\} \leq W(C,v) + \sum_{i<\tau(v)} \max\{w(u,v) \mid u \in I_i \cap N(v)\} + w(v_j, v) = \sigma(v) + w(v_j, v)$. By applying Bound 1, we have $W(F) \leq W(C \cup \{v_j\}) + \sum_{i<\tau(v_j)} \max\{\sigma'(v) \mid v \in I_i \cap N(v_j)\} \leq W(C \cup \{v_j\}) + \sum_{i<\tau(v_j)} \max\{\sigma(v) + w(v_j, v) \mid v \in I_i \cap N(v_j)\}$. Therefore, we can set

$$upper(v_j) = W(C) + W(C, v_j) + \sum_{i<\tau(v_j)} \max\{\sigma(v) + w(v_j, v) \mid v \in I_i \cap N(v_j)\}.$$

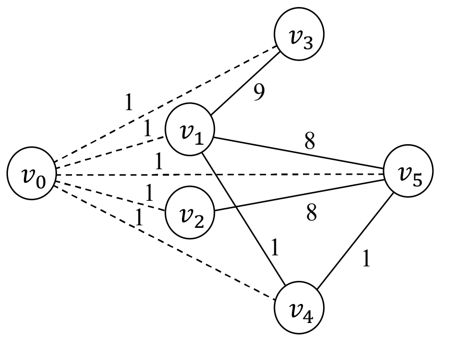

Figure 1: Example graph to illustrate upper bounds

Example Here we give an example to illustrate the new upper bound with the graph G of Figure 1. Suppose $C = \{v_0\}$, $S = \{v_1, v_2, v_3, v_4, v_5\}$, and S is partitioned into $I_1 = \{v_1, v_2\}$, $I_2 = \{v_3, v_4\}$, and $I_3 = \{v_5\}$. Then $\sigma(v_1) = W(C, v_1) = 1$, $\sigma(v_2) = W(C, v_2) = 1$, $\sigma(v_3) = W(C, v_3) + \max\{w(v_1, v_3)\} = 10$, $\sigma(v_4) = W(C, v_4) + \max\{w(v_1, v_4)\} = 2$, $\sigma(v_5) = W(C, v_5) + \max\{w(v_1, v_5), w(v_2, v_5)\} + \max\{w(v_4, v_5)\} = 10$. Bound 1 gives $W(C) + \max\{\sigma(v_1), \sigma(v_2)\} + \max\{\sigma(v_3), \sigma(v_4)\} + \max\{\sigma(v_5)\} = 0+1+10+10 = 21$. For Bound 2, we compute the $upper(\cdot)$ value for each vertex, and get $upper(v_1) = W(C) + W(C, v_1) = 0+1 = 1$, $upper(v_2) = W(C) + W(C, v_2) = 0+1 = 1$, $upper(v_3) = W(C) + W(C, v_3) + \max\{\sigma(v_1) + w(v_1, v_3)\} = 0 + 1 + (1 + 9) = 11$, $upper(v_4) = W(C) + W(C, v_4) + \max\{\sigma(v_1) + w(v_1, v_4)\} = 0 + 1 + (1 + 1) = 3$, and $upper(v_5) = W(C) + W(C, v_5) + \max\{\sigma(v_1) + w(v_1, v_5), \sigma(v_2) + w(v_2, v_5)\} + \max\{\sigma(v_4) + w(v_4, v_5)\} = 0 + 1 + (1 + 8) + (2 + 1) = 13$. So the upper bound obtained is $\max\{upper(v_1), upper(v_2), upper(v_3), upper(v_4), upper(v_5)\} = 13$, which is optimal and corresponds to the clique $C_{max} = \{v_0, v_1, v_4, v_5\}$ in this graph.

An Efficient *GetBranches* Function

In this section, we develop an efficient *GetBranches* function based on the new upper bound above. The pseudocode is shown in Algorithm 3.

Recall that the *GetBranches* function is responsible for partitioning the candidate S into a pruned set P and a branching set B such that expanding the current partial clique C with only the vertices from P will not lead to a clique better than the current best clique C_{max}.

We construct the pruned set P by creating a series of independent sets, and define P to be the union of them. Specifically, suppose we have created $k-1$ maximal independent sets $I_1, I_2, \ldots, I_{k-1}$, and we are now trying adding a vertex to I_k (I_k may be empty or already contain some vertices). We assume that now the set $P := \bigcup_{i=1}^{k} I_i$ satisfies the above-mentioned property for a pruned set. We use T to denote the set of vertices that are not processed, i.e., vertices that are not added to B or any independent set. And we use X to denote the set of vertices that are not processed and not adjacent to any vertex in I_k.

If X is not empty, we select from X the vertex v with the max degree in $G[S]$ to process. It is known that adding vertices in such a way favors the production of large independent sets (San Segundo et al. 2016; San Segundo, Furini, and Artieda 2019).To add v to I_k, we must ensure that the upper bound of $W(F)$ with $C \subseteq F \subseteq C \cup P \cup \{v\}$ is no greater than $W(C_{max})$. For a clique F with $C \subseteq F \subseteq C \cup P \cup \{v\}$, if $v \notin F$, then $C \subseteq F \subseteq C \cup P$, so $W(F) \leq W(C_{max})$ by the definition of P. Therefore, we only need to consider cliques containing v. In the previous section, we have shown that an upper bound for such cliques is $upper(v) = W(C) + W(C, v) + \sum_{i<\tau(v)} \max\{\sigma(u) + w(v, u) \mid u \in I_i \cap N(v)\}$. If $upper(v)$ is no larger than $W(C_{max})$, we add v to I_k (and accordingly to P), compute and store $\sigma(v)$, and remove the neighbors of v from X. Otherwise, we add v to the branching set B. Notice that when computing $upper(v)$, all the $\sigma(\cdot)$ values used are already available, so the computation can be fast.

If X is empty, then no more vertices can be added to the current independent set. If T is not empty, we open a new independent set and repeat the above steps to process the remaining vertices. Otherwise, all vertices have been processed and we return the final branching set B.

Algorithm 3: GetBranches(G, C, S, C_{max})

```
 1: $B \leftarrow \emptyset$; $T \leftarrow S$; $k \leftarrow 0$
 2: while $T \neq \emptyset$ do
 3:     $k \leftarrow k + 1$; $I_k \leftarrow \emptyset$; $X \leftarrow T$
 4:     while $X \neq \emptyset$ do
 5:         $v \leftarrow$ the vertex with max degree in $X$
 6:         $X \leftarrow X \setminus \{v\}$; $T \leftarrow T \setminus \{v\}$
 7:         $upper(v) \leftarrow W(C) + W(C, v) +$
              $\sum_{i<k} \max\{\sigma(u) + w(u, v) \mid u \in I_i \cap N(v)\}$
 8:         if $upper(v) \leq W(C_{max})$ then
 9:             $\sigma(v) \leftarrow W(C, v) +$
                  $\sum_{i<k} \max\{w(u, v) \mid u \in I_i \cap N(v)\}$
10:             $I_k \leftarrow I_k \cup \{v\}$; $X \leftarrow X \setminus N(v)$
11:         else
12:             $B \leftarrow B \cup \{v\}$
13:         end if
14:     end while
15: end while
16: return $B$
```

Running-Time Analysis

This section analyzes the running-time bound of the algorithm MEWCat. We first show the following lemma.

Lemma 1. *For the $|B|$ branches generated at Step 10 in Algorithm 2, in each branch, the size of the set S decreases by at least $|B|$.*

Proof. Let d denote the maximum degree in $G[S]$. When a vertex $v \in S$ is included to C, all vertices not adjacent to v in S (including v itself) are removed from S, so the size of S decreases by at least $|S| - d_{G[S]}(v) \geq |S| - d = |B'|$. After Step 8 in Algorithm 2, $|B'| \geq |B|$, so the lemma holds. $\square$

Based on this lemma, we prove the following theorem.

Theorem 2. *MEWCat solves MEWCP in $O^*(1.4423^n)$ time.*

Proof. It is easy to see that when $S = \emptyset$, MEWCP solves the problem directly. We use $T(s)$ to denote the maximum number of leaves in the search tree generated by MEWCP on any instance with $|S| = s$. We prove that $T(s) = O(3^{\frac{s}{3}})$ by an induction on s. When $s = 1$, apparently there exists a positive real number c such that $T(1) \leq c \cdot 3^{\frac{s}{3}}$. Now suppose that $T(s) \leq c \cdot 3^{\frac{s}{3}}$ for all $s \leq l - 1$. We show it also holds for $s = l$. Note that the only branching operation in MEWCat is Step 10 in Algorithm 2. This operation generates $|B|$ branches, and in each branch, the size of S decreases by at least $|B|$ by Lemma 1. Let l denote the size of S, and l_i denote the size of the new S in the i-th branch. We have that $l - l_i \geq |B|$ for each i. By the assumption, we have that $T(l_i) \leq c \cdot 3^{\frac{l_i}{3}}$. Thus, $T(l) \leq \sum_{i=1}^{|B|} T(l_i) \leq \sum_{i=1}^{|B|} c \cdot 3^{\frac{l_i}{3}} \leq \sum_{i=1}^{|B|} c \cdot 3^{\frac{l-|B|}{3}} = c \cdot 3^{\frac{l}{3}} \cdot |B| \cdot 3^{\frac{-|B|}{3}}$. It can be verified that the function $f(i) = i \cdot 3^{\frac{-i}{3}}$ takes the maximum value 1 at $i = 3$ when restricting i to take integer values. Thus, $T(l) \leq c \cdot 3^{\frac{l}{3}} \cdot |B| \cdot 3^{\frac{-|B|}{3}} \leq c \cdot 3^{\frac{l}{3}}$. Therefore, $T(s) \leq c \cdot 3^{\frac{s}{3}}$ also holds for $s = l$. We always have $T(s) = O(3^{\frac{s}{3}})$. Note that other steps of MEWCat can be executed in polynomial time. Thus, the running time of MEWCat is bounded by $O^*(3^{\frac{n}{3}}) = O^*(1.4423^n)$. $\square$

To the best of our knowledge, MEWCat is the first practical exact solver for MEWCP that has a worst-case time complexity better than the trivial bound $O^*(2^n)$.

Computational Experiments

In this section, we present the results of the experiments carried out to evaluate our new exact solver MEWCat[2]. MEWCat is written in C++, and compiled by g++ 9.4.0 with optimization option -O3. All the experiments are carried out in single-threaded mode on a 64-core Intel(R) Xeon(R) Gold 6226R CPU @ 2.90GHz, equipped with 512 GB of memory and running a 64-bit Ubuntu 20.04 operation system. The time limit to solve each instance is set to 7200 seconds.

We compare MEWCat with two state-of-the-art exact solvers BBEWC (San Segundo et al. 2019) and MECQ (Shimizu, Yamaguchi, and Masuda 2020). The source code of MECQ is publicly available[3]. We compile it by the same configuration as MEWCat. However, the source code of BBEWC is not available, so we scaled the reported results for comparison. Specifically, we run the clique algorithm *dfmax*[4] on the benchmarks graphs r300.5, r400.5, and r500.5. The computing times are 0.17, 0.93, and 3.40 seconds respectively. The results reported in (San Segundo et al. 2019) are 0.19, 1.16, and 4.37 seconds respectively. We compute the factor $(0.17/0.19 + 0.93/1.16 + 3.40/4.37)/3 = 0.82$, and use it to multiply the reported times of BBEWC. Comparing codes run on different machines in this way is common in the literature on clique problems (San Segundo, Rodríguez-Losada, and Jiménez 2011). Note that Hosseinian, Fontes, and Butenko (2020) also proposed an exact algorithm for MEWCP. But that algorithm performs much worse than either BBEWC or MECQ according to the reported results, so it is is not considered in our experiments.

instances	time (seconds)			tree size (millions)	
	MEWCat	MECQ	BBEWC	MEWCat	MECQ
brock400_2	**2659.44**	tle	tle	196.9	inf
brock400_3	**1301.82**	4120.80	tle	78.4	363.6
brock400_4	**743.35**	2158.84	tle	37.1	157.8
DSJC1000.5	**761.14**	1059.02	tle	86.5	200.2
gen200_p0.9_44	**748.29**	tle	4759.73	48.7	inf
gen200_p0.9_55	**9.98**	246.74	147.65	0.3	13.4
hamming8-2	**1.22**	20.71	tle	0.0	0.5
p_hat300-3	**108.02**	tle	tle	9.0	inf
p_hat500-2	**19.01**	2076.05	tle	1.1	175.4
p_hat700-2	**411.37**	tle	tle	21.4	inf
san200_0.7_2	**0.16**	2.43	1128.50	0.0	0.3
san200_0.9_3	**424.82**	5619.84	tle	24.8	510.6
san400_0.5_1	**0.41**	0.51	106.19	0.0	0.0
san400_0.7_1	**1.69**	15.93	tle	0.0	0.5
san400_0.7_2	**6.03**	53.14	tle	0.3	2.8
san400_0.7_3	**24.08**	212.14	tle	1.9	20.6
san400_0.9_1	**2363.83**	tle	tle	25.8	inf
san1000	**4.28**	22.13	tle	0.1	0.3
sanr200_0.9	**3263.35**	tle	tle	287.1	inf
sanr400_0.7	**996.8**	2577.28	tle	129.1	495.6

Table 2: Running times in seconds and tree sizes in millions on DIMACS graphs. The best times are in bold. "tle" means the time limit is exceeded, and the corresponding tree size is marked "inf" (infinite) in this case. Column opt is the weight of the maximum edge-weighted clique.

DIMACS Graphs

The DIMACS graphs are from the 2nd DIMACS implementation challenge[5]. These 80 graphs are standard benchmarks for MCP, MVWCP, and MEWCP (San Segundo, Nikolaev, and Batsyn 2015; Li, Jiang, and Manyà 2017; Fang, Li, and Xu 2016; San Segundo et al. 2019; Shimizu, Yamaguchi, and Masuda 2020). Most instances contain hundreds of vertices, and the density ranges from 0.04 to nearly 1.0. We set

[2]Our Appendix, code, and data are publicly available at the website https://github.com/afkbrb/MEWCat

[3]http://www2.kobe-u.ac.jp/~ky/source/index.html

[4]http://archive.dimacs.rutgers.edu/pub/dsj/clique/

[5]http://dimacs.rutgers.edu/archive/Challenges/

instances	time (seconds)			tree size	
	MEWCat	MECQ	BBEWC	MEWCat	MECQ
socfb-UIllinois	**0.59**	40.66	60.63	895	2.74m
cond-mat-2003	**0.14**	2.15	0.26	3	48
ia-email-EU	0.30	2.59	**0.07**	106	435
rgg_n_2_15_s0	**0.15**	2.30	0.33	14	322
ia-enron-large	**0.44**	2.90	0.52	583	10494
socfb-UF	**1.08**	75.01	511.22	2135	3.76m
socfb-Texas84	**1.14**	274.00	323.86	1997	15.85m
tech-internet-as	0.74	4.44	**0.16**	31	178
cond-mat-2005	**0.21**	3.79	0.45	5	109
sc-nasasrb	**0.58**	8.05	2.95	2564	8652
soc-brightkite	**0.56**	9.66	5.16	456	174180
soc-loc-brightkite	**0.50**	9.49	7.72	112	111158
rgg_n_2_16_s0	**0.30**	10.30	1.33	45	334
soc-themarker	**16.00**	342.23	906.03	118662	12.81m
rec-eachmovie	**11.33**	556.64	30.43	15973	58973
soc-Slashdot0811	**1.05**	16.73	5.03	699	68934
soc-Slashdot0902	**1.11**	19.09	8.00	1046	95550
sc-pkustk11	**1.07**	22.18	8.54	3364	21103
ia-wiki-Talk	**1.60**	23.75	3.22	1698	13917
sc-pkustk13	**1.01**	26.06	14.52	2316	37629

Table 3: Running times in seconds and tree sizes on real-world graphs. The "m" after a number means million.

the weight of each edge $(u, v) \in E$ as $(u + v) \bmod 200 + 1$, the same way as in (San Segundo et al. 2019) and (Shimizu, Yamaguchi, and Masuda 2020).

We show the running times and tree sizes (i.e., the number of tree nodes) in Table 2. Note that, "easy" instances that can be solved in 100 seconds by every solver and "hard" instances that no solver can solve within the given time limit are omitted. As a result, we get 20 instances left in the table (the full results can be found in the Appendix). The tree sizes of BBEWC are omitted in the table due to two reasons: (1) BBEWC counts only internal nodes, and that number can be much smaller than the tree size. (2) BBEWC cannot solve many instances in the given time limit, so the corresponding tree sizes are not available.

In general, MEWCat is significantly faster than the other two solvers for these instances based on the following facts. First, out of the whole 80 DIMACS graphs, the number of instances solved by MEWCat is 52, clearly exceeding the number of instances solved by MECQ and BBEWC, which is 46 and 36, respectively. Second, MEWCat achieves considerable speedups on many instances. For example, the speedups are 15x, 17x, 67x, and 109x for *gen200_p0.9_55*, *hamming8-2*, *p_hat300_3*, and *p_hat500_2*, respectively. Comparing average times for solving the instances in this table, MEWCat is 4.43x faster than MECQ, and is also much faster than BBEWC which cannot even provide results for 16 instances. Notably, MEWCat is the only algorithm that can solve the 5 instances *brock400_2*, *p_hat300_3*, *p_hat700-2*, *san400_0.9_1*, and *sanr200_0.9*.

In terms of tree sizes, MEWCat dominates MECQ for all these instances. This indicates that the new upper bound proposed in this paper is tighter than the one proposed in (Shimizu, Yamaguchi, and Masuda 2020).

Real-World Graphs

The real-world graphs considered here can be found in the Network Repository[6] and are also used in the experiments in (San Segundo et al. 2019) and (Shimizu, Yamaguchi, and Masuda 2020). These graphs are from several fields including social networks, scientific computing networks, and technological networks. The edge weights are assigned in the same way as the DIMACS graphs. These graphs are much larger than the DIMACS graphs and are sparser on the other hand. The number of vertices of these graphs ranges from 30795 to 94893, and the density ranges from 0.0000850 to 0.00267.

The full results are shown in Table 3. It is clear that MEWCat significantly outperforms other solvers on these instances. The speedup of MEWCat over the best of MECQ and BBEWC achieves 69x, 69x, and 240x for *socfb-UIllinois*, *socfb-UF*, and *socfb-Texas84*, respectively. The average time for MEWCat, MECQ, and BBEWC is 2.00, 72.60, and 94.52 seconds, respectively. On average, MEWCat is over 36x and 47x faster than MECQ and BBEWC, respectively.

Random Graphs

We also conduct experiments on random graphs. The random graphs used are generated in the same way as in (San Segundo et al. 2019). An edge between any two vertices exists with a constant probability, and the weight is assigned in the same way as described above.

The results are given in Table 4 in the Appendix due to space limit. We summarize the results below. (1) The average time to solve all these random graphs for MEWCat, MECQ, and BBEWC is 36.43, 92.26, and 84.29 seconds, respectively. The speedup of MEWCat to MECQ and BBEWC is 2.53 and 2.31, respectively. (2) It is observed that, for a fixed $|V|$, as the density increases, the running time of MEWCat increases moderately comparing with the other two solvers. For example, for $|V| = 200$, when the density increases from 0.7 to 0.8, the running time increases 22x, 42x, and 40x for MEWCat, MECQ, and BBEWC, respectively.

Conclusions

We proved that MEWCP is still NP-hard even when the minimum degree of the graph is $n - 2$, in contrast to MVWCP which is polynomial-time solvable when the minimum degree of the graph is at least $n - 3$. This result distinguishes the computational complexity of the two problems for the first time. To address MEWCP, we introduced a new upper bound and developed a practical branch-and-bound solver MEWCat based on it. Theoretically, we proved that MEWCat is guaranteed to find the maximum edge-weighted clique in $O^*(1.4423^n)$ time, while previous solvers cannot guarantee the time complexity to be better than the trivial bound $O^*(2^n)$. We also carried out experiments on different graphs to evaluate the practical performance of MEWCat. The results show that MEWCat significantly outperforms other state-of-the-art exact solvers.

[6]https://networkrepository.com/

Acknowledgments

We thank Tian Bai and Junqiang Peng for discussing about the computational complexity of MEWCP. This work is supported by National Natural Science Foundation of China under grants 62372095, 62172077, and 62372093, Natural Science Foundation of Sichuan Province of China under grants 2023NSFSC0059 and 2023NSFSC1415, and CCF-Huawei Populus Grove Fund - Theoretical Computer Science Special Project.

References

Agapito, L. A.; Fornari, M.; Ceresoli, D.; Ferretti, A.; Curtarolo, S.; and Nardelli, M. B. 2016. Accurate tight-binding Hamiltonians for two-dimensional and layered materials. *Physical Review B*, 93(12): 125137.

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

Chu, Y.; Liu, B.; Cai, S.; Luo, C.; and You, H. 2020. An efficient local search algorithm for solving maximum edge weight clique problem in large graphs. *Journal of Combinatorial Optimization*, 39: 933–954.

Fang, Z.; Li, C.-M.; and Xu, K. 2016. An exact algorithm based on maxsat reasoning for the maximum weight clique problem. *Journal of Artificial Intelligence Research*, 55: 799–833.

Fontes, D. B.; Goncalves, J. F.; and Fontes, F. A. 2018. An evolutionary approach to the maximum edge weight clique problem. *Recent Advances in Electrical & Electronic Engineering (Formerly Recent Patents on Electrical & Electronic Engineering)*, 11(3): 260–266.

Garey, M. R.; Johnson, D. S.; and Stockmeyer, L. 1974. Some simplified NP-complete problems. In *Proceedings of the sixth annual ACM symposium on Theory of computing*, 47–63.

Gary, M. R.; and Johnson, D. S. 1979. Computers and Intractability: A Guide to the Theory of NP-completeness.

Gouveia, L.; and Martins, P. 2015. Solving the maximum edge-weight clique problem in sparse graphs with compact formulations. *EURO Journal on Computational Optimization*, 3(1): 1–30.

Hosseinian, S.; Fontes, D. B.; and Butenko, S. 2018. A nonconvex quadratic optimization approach to the maximum edge weight clique problem. *Journal of Global Optimization*, 72: 219–240.

Hosseinian, S.; Fontes, D. B.; and Butenko, S. 2020. A lagrangian bound on the clique number and an exact algorithm for the maximum edge weight clique problem. *INFORMS Journal on Computing*, 32(3): 747–762.

Karp, R. M. 2010. *Reducibility among combinatorial problems*. Springer.

Li, C.-M.; Fang, Z.; Jiang, H.; and Xu, K. 2018a. Incremental upper bound for the maximum clique problem. *INFORMS Journal on Computing*, 30(1): 137–153.

Li, C.-M.; Jiang, H.; and Manyà, F. 2017. On minimization of the number of branches in branch-and-bound algorithms for the maximum clique problem. *Computers & Operations Research*, 84: 1–15.

Li, R.; Wu, X.; Liu, H.; Wu, J.; and Yin, M. 2018b. An efficient local search for the maximum edge weighted clique problem. *IEEE Access*, 6: 10743–10753.

Li, X.; Wu, M.; Kwoh, C.-K.; and Ng, S.-K. 2010. Computational approaches for detecting protein complexes from protein interaction networks: a survey. *BMC genomics*, 11(1): 1–19.

Ma, T.; and Latecki, L. J. 2012. Maximum weight cliques with mutex constraints for video object segmentation. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, 670–677. IEEE.

Pullan, W. 2008. Approximating the maximum vertex/edge weighted clique using local search. *Journal of Heuristics*, 14: 117–134.

San Segundo, P.; and Artieda, J. 2015. A novel clique formulation for the visual feature matching problem. *Applied Intelligence*, 43: 325–342.

San Segundo, P.; Coniglio, S.; Furini, F.; and Ljubić, I. 2019. A new branch-and-bound algorithm for the maximum edge-weighted clique problem. *European Journal of Operational Research*, 278(1): 76–90.

San Segundo, P.; Furini, F.; and Artieda, J. 2019. A new branch-and-bound algorithm for the maximum weighted clique problem. *Computers & Operations Research*, 110: 18–33.

San Segundo, P.; Lopez, A.; Batsyn, M.; Nikolaev, A.; and Pardalos, P. M. 2016. Improved initial vertex ordering for exact maximum clique search. *Applied Intelligence*, 45: 868–880.

San Segundo, P.; Nikolaev, A.; and Batsyn, M. 2015. Infra-chromatic bound for exact maximum clique search. *Computers & Operations Research*, 64: 293–303.

San Segundo, P.; and Rodriguez-Losada, D. 2013. Robust global feature based data association with a sparse bit optimized maximum clique algorithm. *IEEE Transactions on Robotics*, 29(5): 1332–1339.

San Segundo, P.; Rodríguez-Losada, D.; and Jiménez, A. 2011. An exact bit-parallel algorithm for the maximum clique problem. *Computers & Operations Research*, 38(2): 571–581.

Shimizu, S.; Yamaguchi, K.; and Masuda, S. 2017. Mathematical programming formulation for the maximum edge-weight clique problem. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences (in Japanese)*, 100: 313–315.

Shimizu, S.; Yamaguchi, K.; and Masuda, S. 2019. A branch-and-bound based exact algorithm for the maximum edge-weight clique problem. *Computational Science/Intelligence & Applied Informatics 5*, 27–47.

Shimizu, S.; Yamaguchi, K.; and Masuda, S. 2020. A maximum edge-weight clique extraction algorithm based on branch-and-bound. *Discrete Optimization*, 37: 100583.

Tomita, E.; Akutsu, T.; and Matsunaga, T. 2011. *Efficient algorithms for finding maximum and maximal cliques: Effective tools for bioinformatics*. IntechOpen.

Tomita, E.; and Seki, T. 2003. An efficient branch-and-bound algorithm for finding a maximum clique. In *International conference on discrete mathematics and theoretical computer science*, 278–289. Springer.

Wu, Q.; and Hao, J.-K. 2015. A review on algorithms for maximum clique problems. *European Journal of Operational Research*, 242(3): 693–709.

Xiao, M.; Huang, S.; Zhou, Y.; and Ding, B. 2021. Efficient reductions and a fast algorithm of maximum weighted independent set. In *Proceedings of the Web Conference 2021*, 3930–3940.