A new branch-and-bound algorithm for the maximum edge-weighted clique problem

Pablo San Segundo[1]

Universidad Politécnica de Madrid (UPM), Madrid, Spain
Centre of Automation and Robotics (CAR), CSIC-UPM, Madrid, Spain
pablo.sansegundo@upm.es

Stefano Coniglio

University of Southampton, Southampton, United Kingdom
School of Mathematical Sciences
s.coniglio@soton.ac.uk

Fabio Furini

Université Paris-Dauphine, PSL Research University, CNRS, Paris, France
fabio.furini@dauphine.fr

Ivana Ljubić

ESSEC Business School of Paris, Cergy-Pontoise, France
ivana.ljubic@essec.edu

Abstract

We study the maximum edge-weighted clique problem, a problem related to the maximum (vertex-weighted) clique problem which asks for finding a complete subgraph (i.e., a clique) of maximum total weight on its edges. The problem appears in a wide range of applications, including bioinformatics, material science, computer vision, robotics, and many more. In this work, we propose a new combinatorial branch-and-bound algorithm for the problem which relies on a novel bounding procedure capable of pruning a very large amount of nodes of the branch-and-bound tree. Extensive computational experiments on random and structured graphs, encompassing standard benchmarks used in the literature as well as recently introduced real-world large-scale graphs, show that our new algorithm outperforms the state-of-the-art by several orders of magnitude on many instances.

Keywords: combinatorial optimization, branch-and-bound, maximum edge-weighted clique problem

[1]corresponding author

1. Introduction

A large number of emerging applications in scientific areas including material sciences, bioinformatics, computer vision, and robotics ask for finding a densely connected subgraph of a given graph which maximizes a measure of correlation among its vertices. When the pairwise correlation is represented by the weight of an edge of the input graph, this problem can be modeled as a variant of the maximum (vertex-weighted) clique problem in which the goal is to find a complete subgraph (i.e., a clique) of maximum weight on its edges. This problem, which is known in the literature as the maximum edge-weighted clique problem, is the subject of study of this article.

In material sciences, the task is to find a set of materials with high pairwise similarity modeled as edge weights in a graph where the materials are vertices) which, due to having similar features, can be adopted interchangeably [2]. In bioinformatics, the focus is on extracting protein structures from protein-interaction networks where the vertices are proteins and the (edge) weights model their degree of interaction [3, 4, 23, 44]. A relevant application arising in computer vision and pattern recognition [24, 34] and robotics [35] is found in data correspondence problems, where two sets of elements with different features need to be matched. The problem can be solved by computing a maximum-weight clique in a so-called association graph whose vertices represent pairs of elements, one from each set, and an edge between two pairs indicates that they are compatible. Edge weights in this association graph can provide a quantitative measure of the quality of the corresponding association.

1.1. Max clique problems

Given a simple undirected graph $G = (V, E)$ with $|V|$ vertices and $|E|$ edges, a subset $C \subseteq V$ of vertices is called a *clique* if every pair of vertices in C is connected by an edge in E. The classical *Maximum Clique Problem* (MCP) asks for determining a clique of maximum size.

We focus, in this work, on the edge-weighted case, known in the literature as the *Maximum Edge-Weighted Clique Problem* (MEWCP). Letting $c : E \to \mathbb{R}^+$ be a weight function associating a weight c_e with each edge $e \in E$ of the graph, the MEWCP asks for finding a clique C of G which maximizes the total edge weight of the edges whose endpoints are both contained in C. Formally, this corresponds to maximizing the quantity

$$c(C) := \sum_{e \in E[C]} c_e,$$

where $E[C] := \{e = \{u, v\} \in E : u, v \in C\}$, i.e., the subset of edges in E incident to two vertices in C. We denote by $\omega_c(G)$ the optimal solution value of the MEWCP on G, and call it the *edge-weighted clique number* of G. An illustration is given in Figure 1.

Other generalizations of the MCP typically include the presence of non-unit weights either on the vertices or on both the edges and the vertices. The vertex-weighted case is known as the

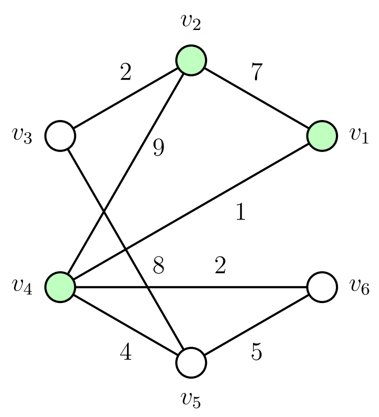

Figure 1: A graph G with $|V| = 6$ vertices and $|E| = 8$ edges. The clique $C = \{v_1, v_2, v_4\}$ (whose vertices are highlighted in green) is the (unique) maximum edge-weighted clique that G contains. The edge-weighted clique number of G is $\omega_c(G) = c(C) = 17$.

Maximum Vertex-Weighted Clique Problem (MVWCP), while the vertex and edge-weighted case is known as the *Maximum Total-Weighted Clique Problem* (MTWCP). See [13] for a recent survey.

1.2. Notation

Throughout the paper, we adopt the following notation. For each $u \in V$, we denote by $N(u) := \{v \in V : \{u, v\} \in E\}$ the *neighborhood* of u. The complement of the graph $G = (V, E)$ is the graph $\overline{G} = (V, \overline{E})$ where $\overline{E} := \{e = \{u, v\} \text{ with } u, v \in V : e \notin E\}$ is the set of *non-edges* of G. A subset of vertices $I \subseteq V$ is an *independent set* (or stable set) if it is a clique of $\overline{G}$. The number of vertices of a largest clique of G (the *clique number* of G) is denoted by $\omega(G)$.

For a subset of vertices $U \subseteq V$, we denote by $G[U] := G(U, E[U])$ the graph *induced* by U, with vertex set U and edge set $E[U]$. $E[U]$ corresponds to the subset of edges in E incident to two vertices in U (i.e., all the edges $e = \{u, v\}$ with $u, v \in U$).

For each vertex $v \in V$, we denote by $\delta(v) \subseteq E$ the subset of edges incident to v. For each $U \subseteq V$, we denote by $\delta(U)$ the subset of edges incident to exactly one endpoint in U (i.e., all the edges $e = \{u, v\}$ with $u \in U,\ v \in V \setminus U$). Furthermore, we use $\delta(u, U)$ to denote the set of edges that are incident to $u \in V$ and any vertex in U, i.e., $\delta(u, U) := \delta(u) \cap \delta(U)$.

1.3. Paper outline and contribution

In this work, we introduce a new combinatorial branch-and-bound algorithm for the MEWCP which employs a new bounding procedure which we specifically design for this problem. The procedure, while based on an Integer Linear Programming (ILP) formulation for the MTWCP, is purely combinatorial, and it operates on a suitably defined vertex and edge-weighted graph.

The new algorithm, which is designed to efficiently perform the branching and bounding operations, significantly outperforms the state-of-the-art on a standard set of benchmark instances used in literature. We also test our algorithm on a benchmark comprising large-scale real-world networks with millions of edges, and show that, even for these challenging graphs, our method allows for finding optimal solutions within a very short computing time.

The paper is structured as follows. Section 2 summarizes the main results in the literature on solving the MEWCP. Section 3 describes the new algorithm in detail, while Section 4 complements the previous section with an illustrative example. Extensive computational results are reported in Section 5. Section 6 presents some concluding remarks and elaborates on future research directions.

2. Literature review

In the literature, many of the studies on edge-weighted clique problems have considered the case where the size of the clique is bounded by a constant k and one has to find a subset of at most k vertices of maximum edge weight. Examples of this are the *maximum diversity problem* and the related *k-subgraph* and *k-cluster problems*. In all these cases, the input graph is usually assumed to be complete. For more information on diversity problems, we refer the reader to [30, 17, 26]. The MEWCP can be reduced to a maximum diversity problem by adding dummy edges with adequate negative weights to make the graph complete and by dropping the cardinality constraint on the clique size by letting $k := |V|$. Exact solution methods for the maximum diversity problem are mainly of branch-and-cut type, see [15, 7, 29, 25, 43].

Exact algorithms in the literature are either based on *ad hoc* branch-and-bound methods or on solving mathematical programming formulations with a state-of-the-art solver. We summarize them in the following.

The authors of [14] propose a quadratic programming formulation for the MEWCP which maximizes a quadratic function over the unit hypercube (as opposed to the well known Motzkin-Straus quadratic formulation, whose feasible region is the standard simplex [27]). Besides characterizing global and local optimality conditions for the formulation and the corresponding underlying structures in the input graph, the paper proposes an upper bound based on the continuous relaxation of the new formulation, and embeds it in an exact branch-and-bound algorithm: `CBQ`.

The authors of [42] describe a purely combinatorial branch-and-bound algorithm for the MEWCP, which is the only exact solution method of this type that we are aware of. The algorithm is called `EWCLIQUE` and it exploits a "Russian-doll" scheme which iteratively computes optimal solutions to subproblems of increasing size (the last of which corresponds to the overall problem to be solved). The solution values obtained by solving the subproblems are stored and used to bound (in linear time) the subproblems that are encountered in deeper levels of the branch-and-bound tree. Similar approaches have been described for the MCP, with a Russian-doll algorithm proposed in [28] and a linear-time bound described in [19].

The authors of [11] describe a number of ILP formulations in different spaces of variables aimed to exploit the presence of nonedges in the instances of the MEWCP, and propose a number of enhancements based on different valid inequalities. As one can see from [14] and [42], `EWCLIQUE` and `CBQ` jointly outperform all the formulations proposed in [11] in terms of computational efficiency.

3. New branch-and-bound algorithm

We present, in this section, the new branch-and-bound algorithm that we propose for solving the MEWCP.

Given a node of the branch-and-bound tree, let $C \subseteq V$ be the clique corresponding to the partial solution associated with it (at the root node, C is the empty set). Let also LB be the value of the incumbent solution. If, at any node, $c(C) > LB$, the incumbent solution and LB are both updated. Let $\hat{V}$ be the set of all the vertices such that, if a vertex $u \in \hat{V}$ is individually added to C, $C \cup \{u\}$ is still a clique. $\hat{V}$ corresponds to the intersection of the neighborhoods of the vertices in C, i.e.:

$$\hat{V} := \bigcap_{v \in C} N(v). \tag{1}$$

The subproblem-graph of a specific branching node corresponds to $G[\hat{V}]$. The branch-and-bound tree is obtained by recursively adding one vertex at a time from $\hat{V}$ to C in an n-ary branching fashion.

To reduce the number of nodes generated by the n-ary branching operation, we employ a bounding technique which we describe in the following sections.

3.1. Weight shifting from the edges to the vertices

Given a node of the branch-and-bound tree, consider the associated (partial) clique C and the corresponding candidate set $\hat{V}$. Since all the vertices in C are already part of the solution, adding u to C, for any $u \in \hat{V}$, yields a direct contribution to the objective function equal to the sum of the weights of all the edges which belong to $\delta(u) \cap \delta(C)$ (i.e., to $\delta(u, C)$). In other words, adding u to C increases the objective function value by the quantity:

$$w_u := \sum_{e \in \delta(u,C)} c_e, \qquad u \in \hat{V}. \tag{2}$$

We refer to w_u as the *vertex weight of u induced by C*. According to (2), the effect of any (partial) clique C is of *shifting* the weight c_e from each edge $e = \{u, v\} \in \delta(u, C)$ to its endpoint $u \in \hat{V}$.

Given C and $\hat{V}$, we construct a vertex and edge-weighted graph $\hat{G} = (\hat{V}, \hat{E})$ with $\hat{E} := E[\hat{V}]$ and vertex weights w defined as in Equation (2). Notice that the weight-shifting operation in Equation (2) assigns a vertex weight only to the vertices in $\hat{V}$, whereas it leaves the weight of every edge $e \in \hat{E}$ unchanged. We remark that, assuming that G is connected and that $c_e > 0$ for all $e \in E$, we have $w_u = 0$ for some $u \in \hat{V}$ if and only if $C = \emptyset$ (which happens only at the root node of the branching tree). See Figure 2 for an illustration.

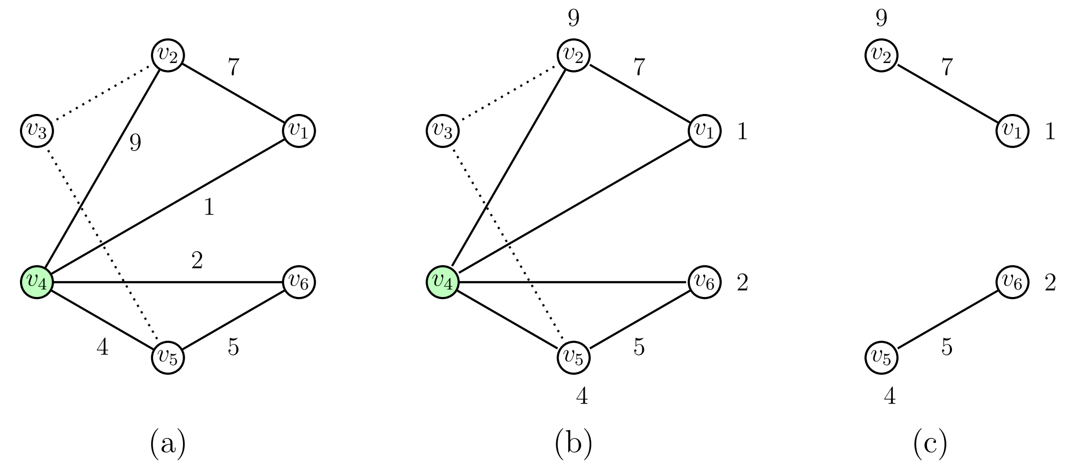

Figure 2: Illustration of the vertex weight induced by C on the nodes $u \in \hat{V}$. (a) The original graph, with an incumbent solution $C = \{v_4\}$ of value 0, highlighted in green, and candidate vertex set $\hat{V} = \{v_1, v_2, v_5, v_6\}$. (b) The original graph with a number associated with each vertex u equal to w_u as defined in Equation (2). Notice that the edge weight of each edge $e \in E[\hat{V}]$ is unchanged. (c) The edge and vertex-weighted graph $\hat{G} = (\hat{V}, \hat{E})$, with the corresponding vertex weight induced by C and the original edge weight on its edges.

We make the following key observation:

Observation. *Let $C \subseteq V$ be a clique in G. Let $\omega_{c,w}(\hat{G})$ denote the optimal solution value of the MTWCP on $\hat{G}$ with edge weights equal to c_e for each $e \in \hat{E}$ and vertex weights defined as in Equation (2) for each $v \in \hat{V}$. Let $\omega_c(G, C)$ be the optimal solution value of the MEWCP on G with the restriction that all the vertices in C be part of the solution. The following holds:*

$$\omega_c(G, C) = c(C) + \omega_{c,w}(\hat{G}).$$

The observation shows that, in our branch-and-bound setting, the MEWCP and the MTWCP are strongly related, as the latter becomes the problem to be solved after any branching operation.

3.2. Constructing the branching and pruned sets B and P

We design our algorithm around a branching scheme which has been successfully adopted in the most effective combinatorial branch-and-bound algorithms proposed for the MCP and the MVWCP, see, e.g., [16, 20, 21, 22, 36, 37, 38, 39, 41]. It relies on the construction of two sets of vertices: the *branching set* B and the *pruned set* P.

At each node of the branch-and-bound tree, we carry out a $|B|$-ary branching operation, thereby creating a tree node per vertex $v \in B$, by individually adding each vertex $v \in B$ to C. The vertices in P are never selected as branching vertices at the current tree node. Therefore, B is always defined as $B := \hat{V} \setminus P$.

The key idea is to construct P in such a way that any subset of its vertices would not suffice, when added to C, to produce a solution of value strictly better than LB. This implies that to improve, if possible, the incumbent, one has to add to C at least one of the vertices in

B. This is precisely what our branch-and-bound algorithm will do as it generates, for each $v \in B$, a child node containing the partial clique $C \cup \{v\}$.

For constructing the pruned set P, we look, in principle, for the largest set P such that:

$$c(C) + \omega_{c,w}(\hat{G}[P]) \leq LB, \tag{3}$$

where $\omega_{c,w}(\hat{G}[P])$ denotes the optimal solution value of the MTWCP for the graph $\hat{G}[P]$ with the original edge weights c_e for all $e \in E[P]$ and vertex weights w_v defined as in Equation (2) for all $v \in P$. This is motivated by the fact that, the larger P, the smaller is the number of vertices in B and, thus, the smaller the number of child nodes created by the branching operation at the current node.

We note that constructing a set P of maximum cardinality which satisfies Inequality (3) is a very hard problem, harder than the MEWCP and complete (in its decision version) for the second level of the polynomial hierarchy:

Proposition. *Finding the largest set P satisfying Inequality* (3) *is $\Sigma_2^{\mathcal{P}}$-hard.*

Proof. Letting $c_e = 0$ for all $e \in \hat{E}$ and $w_v = 1$ for all $v \in \hat{V}$, the decision version of the problem of finding the largest set P satisfying Inequality (3) corresponds to deciding whether, given an integer k, there is a set P of cardinality $|P| \geq k$ such that $w_{c,w}(\hat{G}[P]) \leq LB - c(C)$. Since, due to our choice of edge weights c_e, $\omega_{c,w}(\hat{G}[P]) = \omega(\hat{G}[P])$, this problem is an instance of the *Generalized Node Deletion Problem* (GNDP) which, given a graph $G = (V, E)$ and two integers a and b, asks whether there is a subset $D \subseteq V$ such that $\omega(\hat{G}[\hat{V} \setminus D]) \leq a$ and $|D| \leq b$. Letting $P := \hat{V} \setminus D$, this is the same as asking whether there is a subset $P \subseteq \hat{V}$ such that $\omega(\hat{G}[P]) \leq a$ and $|P| \geq |V| - b$. Since [31] shows that the GNDP is $\Sigma_2^{\mathcal{P}}$-complete, we conclude that finding the largest P which satisfies Inequality (3) is $\Sigma_2^{\mathcal{P}}$-hard. $\square$

We note that a similar results also applies to the problem of finding P which is considered in many of the papers we cited on the MCP and MWCP—to the best of our knowledge, the result in the proposition has not been observed before.

We remark that not only, as the proposition shows, finding the largest P satisfying Inequality (3) is $\Sigma_2^{\mathcal{P}}$-hard but, given any P, even checking whether P satisfies Inequality (3) is $\mathcal{NP}$-hard, as it requires to compute $\omega_{c,w}(\hat{G}[P])$, which is at least as hard as solving the MEWCP. This shows that the decision version of the problem of finding the largest P belongs to $\Sigma_2^{\mathcal{P}}$ and, therefore, that this decision problem is $\Sigma_2^{\mathcal{P}}$-complete.

For these reasons, in Subsection 3.3 we introduce an upper bound on $\omega_{c,w}(\hat{G}[P])$ which, as shown in Subsection 3.4, can be used to build P incrementally in a computationally efficient way.

We remark that, the fact that a vertex v may not belong to P at a certain branch-and-bound node does not imply that the same vertex will belong to P in all the nodes that will descend from it—that vertex could still be part of B at another node of the branch-and-bound tree and, thus, it could still be part of the solutions associated with deeper levels of the implicit-enumeration tree.

We remark that, with this branching scheme, a node of the branch-and-bound tree is fathomed only if $P = \hat{V}$ (i.e., only if $B = \emptyset$), which, since P satisfies Inequality (3) by construction, implies that no solution better than the incumbent can be produced starting from the partial solution C.

3.3. Towards a combinatorial upper bound

We propose a combinatorial way of constructing an upper bound on $\omega_{c,w}(\hat{G}[P])$ for a given $P \subseteq \hat{V}$, which we will then employ (as shown in the next subsection) for the construction of a set P which satisfies Inequality (3).

Our bound is based on observing that, by dividing and assigning the weight of each edge to its endpoints, we obtain an instance of the MVWCP with the property that any bound valid for it is also valid for the original instance of the MEWCP. In our algorithm, we will rely on the independent-set bound for the MVWCP used in many of the state-of-the-art methods for the problem (see, e.g., [8], [16]). The crucial aspect of our procedure is performing the distribution of edge weights in a dynamic fashion, while the partition into independent sets is being constructed.

This section shows the validity of the this bound by drawing its connection to LP-duality theory. Indeed, our bound can be obtained by building a feasible solution to the dual of the Linear Programming (LP) relaxation of an ILP formulation for the MTWCP which we now introduce. Let, for each vertex $v \in P$, the binary variable x_v take value 1 if and only if vertex v belongs to the chosen clique. Let also, for each edge $e = \{u, v\} \in \hat{E}[P]$, the binary variable y_e take value 1 if and only if both endpoints u and v of e belong to the chosen clique. Letting $\mathscr{I}$ be the set of all (maximal) independent sets in $\hat{G}[P]$, a valid formulation for the MTWCP is:

$$\max \sum_{e \in \hat{E}[P]} c_e\, y_e + \sum_{u \in P} w_u x_u \tag{4}$$

$$\sum_{v \in I} x_v \leq 1 \qquad I \in \mathscr{I} \tag{5}$$

$$y_e \leq x_u \qquad e = \{u, v\} \in \hat{E}[P] \tag{6}$$

$$y_e \leq x_v \qquad e = \{u, v\} \in \hat{E}[P] \tag{7}$$

$$x_u \in \{0, 1\} \qquad u \in P \tag{8}$$

$$y_e \in \{0, 1\} \qquad e \in \hat{E}[P]. \tag{9}$$

The objective function (4) corresponds to the total edge and vertex weight of the chosen clique. Constraints (5) impose that at most one vertex be selected from each independent set. Constraints (6)–(7) guarantee $y_e = 1$ if and only if both $x_u = 1$ and $x_v = 1$, for all $e = \{u, v\} \in \hat{E}[P]$.

Note that, due to the equivalence between optimization and separation [12], even solving the LP relaxation of Formulation (4)–(9) is $\mathcal{NP}$-hard. This is because the formulation features exponentially many constraints of type (5), one for each independent set $I \in \mathscr{I}$, and the separation of such constraints is $\mathcal{NP}$-hard (as it corresponds to solving the maximum weighted independent set problem).

What we propose here is a method for computing an upper bound on the optimal solution value of Formulation (4)–(9) for a suitably defined collection of independent sets $\tilde{\mathscr{I}} \subseteq \mathscr{I}$ forming a partition of the vertices of P. The way $\tilde{\mathscr{I}}$ is constructed is explained in the next subsection. By replacing $\mathscr{I}$ by $\tilde{\mathscr{I}}$ in Constraints (5), we obtain a relaxation of the original problem.

The bound that we consider is based on computing a feasible solution to the dual of the LP relaxation of Formulation (4)–(9) with $\mathscr{I}$ replaced by $\tilde{\mathscr{I}}$. Let, for each $I \in \tilde{\mathscr{I}}$, $\pi_I \geq 0$ be the dual variable of Constraints (5) and let, for each $e = \{u, v\} \in \hat{E}[P]$, $\rho_e^u \geq 0$ and $\rho_e^v \geq 0$ be the dual variables of Constraints (6) and (7). Since $\tilde{\mathscr{I}}$ is a partition of P, for each $u \in P$ there is a unique independent set $I(u)$ covering it. With $\mathscr{I}$ restricted to $\tilde{\mathscr{I}}$, the dual of the LP relaxation of Formulation (4)–(9) reads:

$$\min \sum_{I \in \tilde{\mathscr{I}}} \pi_I \tag{10}$$

$$\rho_e^u + \rho_e^v \geq c_e \qquad e = \{u, v\} \in \hat{E}[P] \tag{11}$$

$$\pi_{I(u)} \geq w_u + \sum_{e \in \delta(u,P)} \rho_e^u \qquad u \in P \tag{12}$$

$$\pi_I \geq 0 \qquad I \in \tilde{\mathscr{I}} \tag{13}$$

$$\rho_e^u, \rho_e^v \geq 0 \qquad e = \{u, v\} \in \hat{E}[P]. \tag{14}$$

The objective function (10) is equal to the sum of the values of the π variables. Constraints (11) impose that the weight c_e of each edge $e \in \hat{E}[P]$ be completely covered by the associated ρ_e^u and ρ_e^v variables. For each vertex $u \in P$, Constraints (12) impose that the value of the variable $\pi_{I(u)}$ associated with the independent set $I(u)$ containing u be at least as large as the sum of the values of the ρ variables associated with edges in the star $\delta(u, P)$ of u, plus the weight w_u of the vertex u itself.

Assume that we have chosen values $\tilde{\rho}$ for the ρ variables such that Constraints (11) are satisfied as equations, namely:

$$\tilde{\rho}_e^u + \tilde{\rho}_e^v = c_e, \qquad e = \{u, v\} \in \hat{E}[P]. \tag{15}$$

This choice is w.l.o.g., as each of Constraints (11) is satisfied as an equation in any optimal solution to Formulation (10)–(14). The way $\tilde{\rho}$ is constructed will be explained in the next subsection. For this choice of $\tilde{\rho}$, the weight of each edge $e = \{u, v\} \in \hat{E}[P]$ is completely shared by its endpoints u (via $\tilde{\rho}_e^u$) and v (via $\tilde{\rho}_e^v$). For each vertex $u \in P$, this results in the following vertex weight:

$$\tilde{w}_u := w_u + \sum_{e \in \delta(u,P)} \tilde{\rho}_e^u \qquad u \in P, \tag{16}$$

where w_u corresponds to the original vertex weight shifted from C.

Since $\tilde{\mathscr{I}}$ is a partition of P, in any optimal solution each of the π variables takes the following value, which can be computed in linear time:

$$\tilde{\pi}_I := \max_{u \in I} \{\tilde{w}_u\} \qquad I \in \tilde{\mathscr{I}}. \tag{17}$$

We define $\tilde{\pi}_I := 0$ whenever $I = \emptyset$.

For the given $\tilde{\mathscr{I}}$, the pair $(\tilde{\pi}, \tilde{\rho})$ yields a valid solution to Formulation (10)–(14). With it, we obtain the following upper bound on $\omega_{w,c}(\hat{G}[P])$:

$$UB(P, \tilde{\mathscr{I}}, \tilde{\rho}) := \sum_{I \in \tilde{\mathscr{I}}} \tilde{\pi}_I \geq \omega_{w,c}(\hat{G}[P]). \tag{18}$$

Figure 3 illustrates the computation of the bound for an example graph.

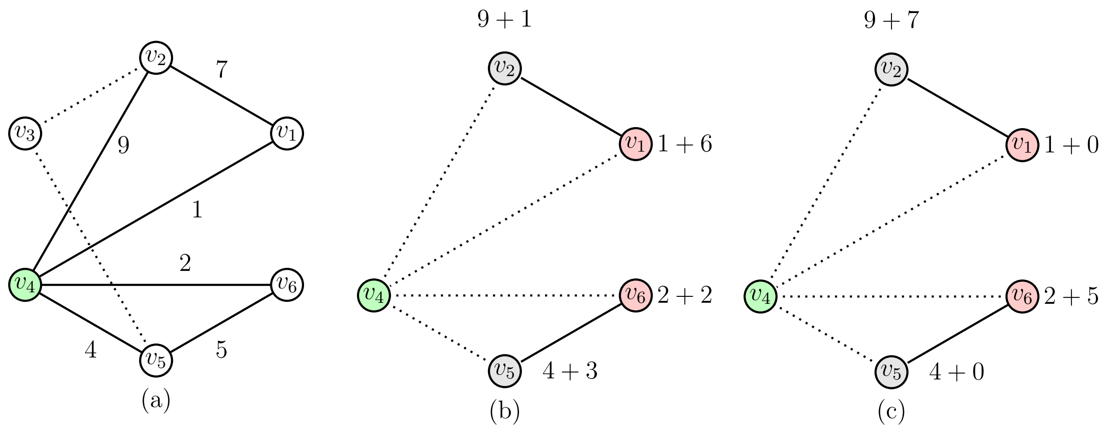

Figure 3: Illustration of the computation of the upper bound $UB(P, \tilde{\mathscr{I}}, \tilde{\rho})$. (a) The original graph, with an incumbent solution $C = \{v_4\}$ of value 0, highlighted in green, and candidate vertex set $\hat{V} = \{v_1, v_2, v_5, v_6\}$. We assume $P = \hat{V}$. (b) The number associated with each vertex u is equal to $\hat{w}_u$, i.e., to the sum of w_u and $\sum_{e \in \delta(u,P)} \tilde{\rho}^u_e$ (the two terms are reported with a '+' symbol between them). The collection $\tilde{\mathscr{I}}$ consists of two independent sets $I_1 = \{v_1, v_6\}$ (in red) and $I_2 = \{v_2, v_5\}$ (in gray), with $\tilde{\pi}_{I_1} = \max\{7, 4\} = 7$, $\tilde{\pi}_{I_2} = \max\{10, 7\} = 10$. The upper bound $UB(P, \tilde{\mathscr{I}}, \tilde{\rho})$ is equal to $\tilde{\pi}_{I_1} + \tilde{\pi}_{I_2} = 17$ (also equal to the optimal solution value). (c) A different choice of $\tilde{\rho}$ gives the upper bound $UB(P, \tilde{\mathscr{I}}, \tilde{\rho}) = \tilde{\pi}_{I_1} + \tilde{\pi}_{I_2} = 7 + 16 = 23$.

As it is clear, the quality of the bound is determined by the choice of $\tilde{\mathscr{I}}$ and $\tilde{\rho}$. The choice we make is described in the next subsection, together with the way we build P.

3.4. Combinatorial procedure for constructing the pruned set P while distributing the edge weights

We now describe the combinatorial procedure that we propose for constructing a pruned set P satisfying Inequality (3). We refer to the overall procedure as `BUILD_P`. The pseudocode is provided in Algorithm 1.

The idea of our procedure is building P incrementally while constructing a collection of independent sets $\tilde{\mathscr{I}} = \{I_1, \ldots, I_{|\tilde{\mathscr{I}}|}\}$ into which P is partitioned (that is, $P = \bigcup_{i=1}^{|\tilde{\mathscr{I}}|} I_i$) and distributing the edge weights so that P satisfies:

$$UB(P, \tilde{\mathscr{I}}, \tilde{\rho}) \leq LB - c(C). \tag{19}$$

Note that Inequality (19) implies that Inequality (3) is satisfied—which, in turn, implies that P is a valid pruned set.

We start from $P = \emptyset$, $\tilde{\mathscr{I}} = \emptyset$, and $\rho_e^u := 0$ for all $u \in \hat{V}$ and $e \in \hat{E}$.

Our procedure works by creating a new independent set I_k at each iteration k and, within that iteration, trying to add to I_k as many vertices in $\hat{V} \setminus P$ as possible (note that, since $P = \bigcup_{i=1}^{|\tilde{\mathscr{I}}|} I_i$, all such vertices are also added to P). Once I_k cannot be enlarged anymore, we consider it *closed* and add it to $\tilde{\mathscr{I}}$. A new empty independent set is then initialized. The procedure is repeated as long as there are still vertices in $\hat{V} \setminus P$. If, at the last iteration, the current independent set is empty, we discard it. Whenever a vertex is added to I_k, we enter the ABSORB and DESORB phases (explained in Sections 3.4.1 and 3.4.2), in which we update the value of the ρ variables and, with them, the edge weight distribution.

Let k be the current iteration and let $I_k := \emptyset$ be the newly created empty independent set. We iterate over all the vertices $u \in \hat{V} \setminus P$ with the property that $I_k \cap N(u) = \emptyset$ (i.e., that $I_k \cup \{u\}$ is an independent set). Given a vertex u, we compute the following value:

$$\texttt{budget}(u) := LB - c(C) - \sum_{i=1}^{k-1} \tilde{\pi}_{I_k} - \tilde{w}_u, \tag{20}$$

where $\tilde{w}_u$ is set as in Equation (16). By construction, adding u to I_k and P would violate Inequality (3) if and only if $\texttt{budget}(u) < 0$. Thus, vertex u is discarded (by removing it from $\hat{V}$ and adding it to B) if $\texttt{budget}(u)$ is strictly smaller than 0, whereas, if $\texttt{budget}(u) \geq 0$, we let:

$$P = P \cup \{u\} \qquad \text{and} \qquad I_k = I_k \cup \{u\}.$$

Then, the ABSORB phase takes place.

3.4.1. ABSORB phase

In the ABSORB phase for a vertex u just added to P and I_k, we determine the value of the ρ_e^u variables for all the edges $e = \{u, v\} \in \delta(u, \hat{V} \setminus P)$ by *absorbing* as much edge weight c_e as possible—that is, we try to cover as much of c_e as possible via ρ_e^u (rather than ρ_e^v) without exceeding $\texttt{budget}(u)$.[2] This phase *preventatively* absorbs the weight of each edge $e = \{u, v\} \in \delta(u, \hat{V} \setminus P)$ so that, if the other endpoint v not in P were to be added to P in a future iteration, the weight c_e of the edge would already be covered.

We assume that the edges are ordered as $e_1, e_2, \ldots, e_{|\delta(u, \hat{V} \setminus P)|}$ (the details on the specific order that we adopt are given in Section 3.5). Let $\xi(u) := |\delta(u, \hat{V} \setminus P)|$ and let $\ell + 1$ be

[2]The value of the ρ_e^u variables for each edge $e = \{u, v\} \in \delta(u, P)$ is set implicitly by the vertices that were added to P in earlier iterations.

the index of the (first, in the order we consider) edge such that $\sum_{j=1}^{\ell} c_{e_j} \leq \texttt{budget}(u)$ and $\sum_{j=1}^{\ell+1} c_{e_j} > \texttt{budget}(u)$. The weights of the edges in $e \in \delta(u, \hat{V} \setminus P)$ are absorbed as follows:

$$\tilde{\rho}^u_{e_j} := \begin{cases} c_{e_j} & \text{if } 1 \leq j \leq \ell \\ \texttt{budget}(u) - \sum_{j=1}^{\ell} c_{e_j} & \text{if } j = \ell + 1 \\ 0 & \text{otherwise} \end{cases} \qquad e_j = \{u, v\} \in \{e_1, \ldots, e_{\xi(u)}\}. \tag{21}$$

After that, the value of $\tilde{w}_u$ is updated as follows:

$$\tilde{w}_u := w_u + \sum_{e \in \delta(u,P)} \tilde{\rho}^u_e + \sum_{e \in \delta(u,\hat{V} \setminus P)} \tilde{\rho}^u_e. \tag{22}$$

Observe that, by construction, the redefined value of $\tilde{w}_u$ in Equation (22) still satisfies the condition $\texttt{budget}(u) \geq 0$. Furthermore, notice that the contribution of the two summation terms in Equation (22) depends on when $\tilde{w}_u$ is updated: before entering P (and, therefore, before the ABSORB phase takes place), it is the vertex weight "imposed by P" that implicitly defines $\tilde{w}_u$ as $w_u + \sum_{e \in \delta(u,P)} \tilde{\rho}^u_e$, whereas the second summation is zero. Once u joins P, the absorption of the weights of the edges $e = \{u, v\} \in \delta(u, \hat{V} \setminus P)$ modifies the entries of the second summation, whereas the weights on the edges between u and P (that is, of the edges $e = \{u, v\} \in \delta(u, P)$), remains unaltered.

Since the values of ρ are always chosen so to satisfy Equation (11), when computing the values of $\tilde{\rho}^u_e$ according to Equation (21) we set $\tilde{\rho}^v_e := c_e - \tilde{\rho}^u_e$ for all the end points v in $N(u) \cap \hat{V} \setminus P$ opposite of u. Namely:

$$\tilde{\rho}^v_{e_j} := \begin{cases} 0 & \text{if } 1 \leq j \leq \ell \\ c_{e_j} - (\texttt{budget}(u) - \sum_{j=1}^{\ell} c_{e_j}) & \text{if } j = \ell + 1 \\ c_{e_j} & \text{otherwise} \end{cases} \qquad e_j = \{u, v\} \in \{e_1, \ldots, e_{\xi(u)}\}. \tag{23}$$

For all $\tilde{\rho}^v_e > 0$, this corresponds to *pushing* the weight c_e (in parts or entirely) to v.

After an independent set I_k has been closed (as no more vertices can be added to it), we update $\tilde{\pi}_{I_k}$ according to Equation (17).

Since, by design of the ABSORB phase, the weight $\tilde{w}_u$ of each vertex $u \in I_k$ can increase only by $\texttt{budget}(u)$, the phase guarantees that Inequality (3), which was satisfied before ABSORB began, is still satisfied after the phase is over.

3.4.2. DESORB phase

After closing an independent set I_k, we carry out the DESORB phase to reduce the difference between the first and the second largest vertex weights among the vertices in I_k. This is done to reduce the value of $\tilde{\pi}_{I_k}$ (which, due to Equation (17), is equal to the weight of the heaviest vertex), as it leads to a larger value of $\texttt{budget}(u)$ for the vertices in $u \in \hat{V} \setminus P$, thus allowing for (potentially) adding more of them to P in future iterations.

Let v_1 and v_2 be the two vertices of largest and second-largest weight in I_k and let Δ be their difference. Namely:

$$v_1 := \arg\max_{v \in I_k}\{\hat{w}_v\} \qquad v_2 := \arg\max_{v \in I_k \setminus \{v_1\}}\{\hat{w}_v\} \qquad \Delta := \hat{w}_{v_1} - \hat{w}_{v_2}. \tag{24}$$

We now iterate over all the edges $e \in \delta(v_1, \hat{V} \setminus P)$, *desorbing* a total of Δ units of the edge weights that v_1 has absorbed. Let $\ell + 1$ be the index of the first (in the order we use) edge such that $\sum_{j=1}^{\ell} c_{e_j} \leq \Delta$ and $\sum_{j=1}^{\ell+1} c_{e_j} > \Delta$. We update $\tilde{\rho}$ as follows:

$$\tilde{\rho}_{e_j}^{v_1} := \begin{cases} 0 & \text{if } 1 \leq j \leq \ell \\ c_{e_j} - \left(\Delta - \sum_{j=1}^{\ell} c_{e_j}\right) & \text{if } j = \ell + 1 \\ c_{e_j} & \text{otherwise} \end{cases} \qquad e_j = \{v_1, u\} \in \{e_1, \ldots, e_{\xi(v_1)}\}. \tag{25}$$

Since the values of ρ are always chosen so to satisfy Equation (11), we set $\tilde{\rho}_e^u := c_e - \tilde{\rho}_e^{v_1}$ for all the end points u in $N(v_1) \cap \hat{V} \setminus P$ opposite of v_1. Namely:

$$\tilde{\rho}_{e_j}^{u} := \begin{cases} c_{e_j} & \text{if } 1 \leq j \leq \ell \\ \Delta - \sum_{j=1}^{\ell} c_{e_j} & \text{if } j = \ell + 1 \\ 0 & \text{otherwise} \end{cases} \qquad e_j = \{v_1, u\} \in \{e_1, \ldots, e_{\xi(v_1)}\}. \tag{26}$$

For all $\tilde{\rho}_e^u > 0$, this corresponds to *pushing* the weight c_e (in parts or entirely) to u. Lastly, the value of $\tilde{w}_u$ is updated as in Equation (22).

3.5. Outline of the overall branch-and-bound algorithm

We refer to the recursive procedure which implements our combinatorial branch-and-bound algorithm as `BBWEC`. The procedure takes as input the current (partial) clique C, the corresponding candidate set $\hat{V}$, and the branching set B. The pseudocode of is given in Algorithm 2.

The algorithm processes the vertices from $v \in B$ iteratively by looking one step ahead. Note that, due to this look-ahead, the number of recursive calls is always smaller than the number of branch-and-bound nodes. It updates C with the inclusion of vertex v (Step 2) and checks if the node of the tree corresponding to the new clique C would be a leaf node (Step 3). If this is the case, the value $c(C_{\max})$ of the global solution $C_{\max}$ is compared to $c(C)$ for a possible update (Step 4).

If we do not get a leaf node, we update $\hat{V}$ as a consequence of the update of C (Step 8), compute P via the procedure `BUILD_P` (Step 9) and construct B (Step 10). Finally, we call `BBEWC` recursively (if B is nonempty) with input C, $\hat{V}$, and B (Step 12). If B is empty (Step 11), no recursive call takes place—this corresponds to fathoming the current node of the branch-and-bound tree.

Once the vertex v has been taken into account for extending C, it is removed from $\hat{V}$ (Step 15). This avoids redundant tree nodes from being generated, further reducing the size of the branch-and-bound tree.

Algorithm 1: `BUILD_P`$(C, \hat{V}, LB)$

Input : A clique C, the corresponding candidate set $\hat{V}$, and LB

Output : The pruned set P

```
$k \leftarrow 0$, $\tilde{\mathscr{I}} \leftarrow \emptyset$, $P \leftarrow \emptyset$, $B \leftarrow \emptyset$, $\tilde{\rho} \leftarrow 0$
while $P \cup B \neq \hat{V}$ do
    $k \leftarrow k+1$, $I_k \leftarrow \emptyset$, $\tilde{\pi}_{I_k} \leftarrow 0$
    foreach $u \in \hat{V} \setminus (P \cup B)$ such that $N(u) \cap I_k = \emptyset$ do
        $\tilde{w}_u \leftarrow w_u + \sum_{e \in \delta(u,P)} \tilde{\rho}^u_e$
        Compute budget$(u)$ as in Eq. (20)
        if budget$(u) \geq 0$ then
            $P \leftarrow P \cup \{u\}$
            $I_k \leftarrow I_k \cup \{u\}$
            Update $\tilde{\rho}$ as in Eq. (21)–(23)                    // ABSORB
            $\tilde{w}_u \leftarrow \tilde{w}_u + \sum_{e \in \delta(u,\hat{V} \setminus P)} \tilde{\rho}^u_e$
        else
            $B \leftarrow B \cup \{u\}$
        end
    end
    Update $\tilde{\pi}_{I_k}$ as in Eq. (17)                        // $I_k$ is closed
    if $|I_k| \geq 2$ then
        Update $\tilde{\rho}$ as in Eq. (25)–(26)                    // DESORB
        Update $\tilde{w}_u$ and $\tilde{\pi}_{I_k}$ as in Eq. (22) and Eq. (17), resp.
    end
    if $I_k \neq \emptyset$ then
        $\tilde{\mathscr{I}} \leftarrow \tilde{\mathscr{I}} \cup \{I_k\}$
    end
end
return $P$
```

Algorithm 2: BBEWC(C, $\hat{V}$, B)

Input : A clique C, the corresponding candidate set $\hat{V}$, and the branching set B.

```
// $C_{\max}$ is a global variable.
for $v \in B$ do
    $C \leftarrow C \cup \{v\}$
    if $\hat{V} \cap N(v) = \emptyset$ then
        if $c(C) > c(C_{\max})$ then
            $C_{\max} \leftarrow C$            // leaf node improving the incumbent $C_{\max}$
        end
    else
        $\hat{V}' \leftarrow \hat{V} \cap N(v)$
        $P \leftarrow$ BUILD_P($C, \hat{V}', c(C_{\max})$)
        $B' \leftarrow \hat{V}' \setminus P$
        if $B' \neq \emptyset$ then
            BBEWC($C, \hat{V}', B'$)            // recursive call
        end
    end
    $\hat{V} \leftarrow \hat{V} \setminus \{v\}$
    $C \leftarrow C \setminus \{v\}$
end
```

Our algorithm starts with the call BBEWC($\emptyset, \hat{V}, \hat{V}$). Before issuing it, we create a heuristic solution C_H via an adaptation to the MEWCP of the Tabu Search heuristic AMTS proposed in [45] for the MVWCP. The solution C_H is produced by, first, evenly shifting the edge weight of each edge to its end points and, then, calling AMTS on the resulting vertex-weighted graph. The corresponding value LB is obtained by recovering the edge weights from C_H.

During the execution of our branch-and-bound algorithm, the global variable $C_{\max}$ corresponds to the best solution found until now. Upon the termination of BBEWC, $C_{\max}$ contains a clique of maximum edge weight.

Throughout the execution of our algorithm, the vertices of G are always sorted in nonincreasing order of degree. Letting V_O be a copy of the original set of vertices, the order is determined by, for each i from n to 1, selecting from V_O a vertex v with minimum degree in the induced subgraph $G[V_O]$, placing it in position i, and removing it from V_O. A similar order is used in many state-of-the art solvers for the MCP, see [37, 38].

Vertices $u \in \hat{V} \setminus (P \cup B)$ in Step 4 of BUILD_P are examined in this order, and the same applies to the edges that are incident to u during the ABSORB and DESORB phases in the calculations/updates of the values of the ρ variables. Differently, in Step 1 of BBEWC the vertices in B are examined in reverse order.

Our efficient implementation of BBEWC relies on a bitstring representation to encode the input graph G as well as the graph $\hat{G}$ that is constructed along the branching tree. This allows for a fast computation of the set $\hat{V}$ (by intersecting bitstrings corresponding to the neighborhoods of the vertices in C) as well as for determining the collection of independent sets $\tilde{\mathscr{I}}$ efficiently.

4. Demonstration of our algorithm

In this section, we provide a demonstration of the execution of our novel BBEWC algorithm on the graph depicted in Figure 1, with 6 vertices and 8 edges. For simplicity, we assume that the order of the vertices is $v_1, v_2, v_3, v_4, v_5, v_6$. The tree is explored in depth-first fashion.

The full branch-and-bound tree is reported in Figure 4. For each of the 7 explored nodes, the associated box reports the sets $\hat{V}$, B, P, and C, together with the edge weight of the clique $c(C)$, the current LB, and the associated upper bound $UB(P, \tilde{\mathscr{I}}, \tilde{\rho})$. For reasons of space, we denote it simply by $UB(P)$. An illustration of how the branching and the pruned sets B and P are obtained for the root and for the first node of the tree is given in Figure 5.

As initial heuristic solution, we consider the clique $C_H = \{v_1, v_2\}$ of weight $c(C_H) = 7$. The initial lower bound LB is, therefore, 7.

At node 3, the incumbent solution is improved and the LB is set to 11 ($C = \{v_4, v_5, v_6\}$). At node 5, the incumbent solution is further improved and the LB is set to 17 ($C = \{v_1, v_2, v_4\}$). This clique is the optimal one. Nodes 3 and 5 correspond to two leaf nodes of the tree, while node 6 is fathomed since the branching set B is empty. The total number of recursive calls is 4 (the calls take place at the root node and at nodes 1, 2, and 4).

We now illustrate in detail how the branching and pruned sets B and P are obtained for the root and for the first node of the tree. An illustration is provided in Figure 5.

4.1. Construction of B and P for the root node

The set of vertices $\hat{V}$ is $\{v_1, v_2, v_3, v_4, v_5, v_6\}$ ($C = \emptyset$). Since $C = \emptyset$, the weight $\tilde{w}_u$ of all the vertices is equal to 0 initially.

The empty independent set I_1 is initialized (colored in red). Every vertex u in the graph has $\texttt{budget}(u) = LB - c(C) = 7$. Therefore, every vertex can enter I_1. Considering the vertices in order, we add vertex v_1 to I_1, which absorbs the full weight (7) of edge $\{v_1, v_2\}$ and 0 units of that of edge $\{v_1, v_4\}$. Then, vertex v_3 is added to I_1, absorbing the full weight (2) of edge $\{v_2, v_3\}$ and 5 units of that of $\{v_3, v_5\}$. Finally, vertex v_6 is added to I_1, absorbing the full weight (2) from edge $\{v_4, v_6\}$ and the full weight (5) from edge $\{v_5, v_6\}$. Since no more vertices form an independent set with I_1, I_1 is closed. We let $\tilde{\pi}_{I_1} := \max\{7, 7, 7\} = 7$.

No desorbtion can take place, as the first and the second heaviest vertices in I_1 have the same weight (i.e., $\Delta = 0$).

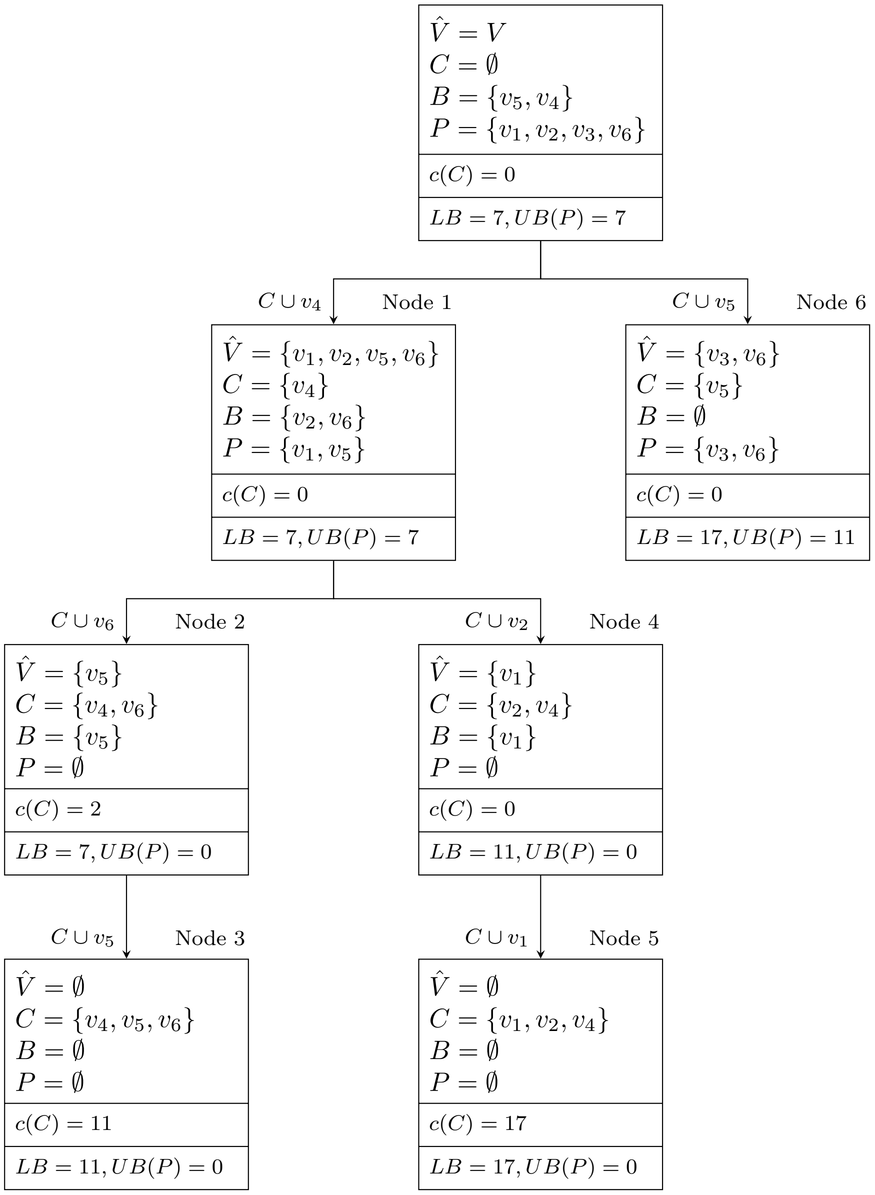

Figure 4: Branch-and-bound tree produced by the BBEWC algorithm for the graph of Figure 1.

We continue, creating the next independent set I_2 (colored in gray), initially empty. Considering the vertices in order, we analyse v_2 which, due to having budget(v_2)= $LB - c(C) - \tilde{\pi}_{I_1} - \tilde{w}_{v_2} = 7 - 0 - 7 - 0 = 0$, enters I_2 but cannot absorb any weight. The 9 units of edge $\{v_2, v_4\}$ are therefore pushed to vertex v_4. The only vertex that can enter I_2 is v_5. Since, though, its weight is $\tilde{w}_{v_5} = 3$, it has negative budget (budget(v_5) $= 7 - 0 - 7 - 3 = -3$). Therefore, v_5 does not enter I_2 and I_2 is closed. Since $|I_2| < 2$, no desorbing takes place.

A new independent set I_3 is created. Since the only vertex that can enter it is v_4 but it has negative budget, I_3 remains empty and gets discarded.

BUILD_P terminates, producing $P = \{v_1, v_2, v_3, v_6\}$, with $UB(P) = 7$ and $B = \{v_5, v_4\}$. We remark that the order in which the vertices are listed in B corresponds to the reverse order in

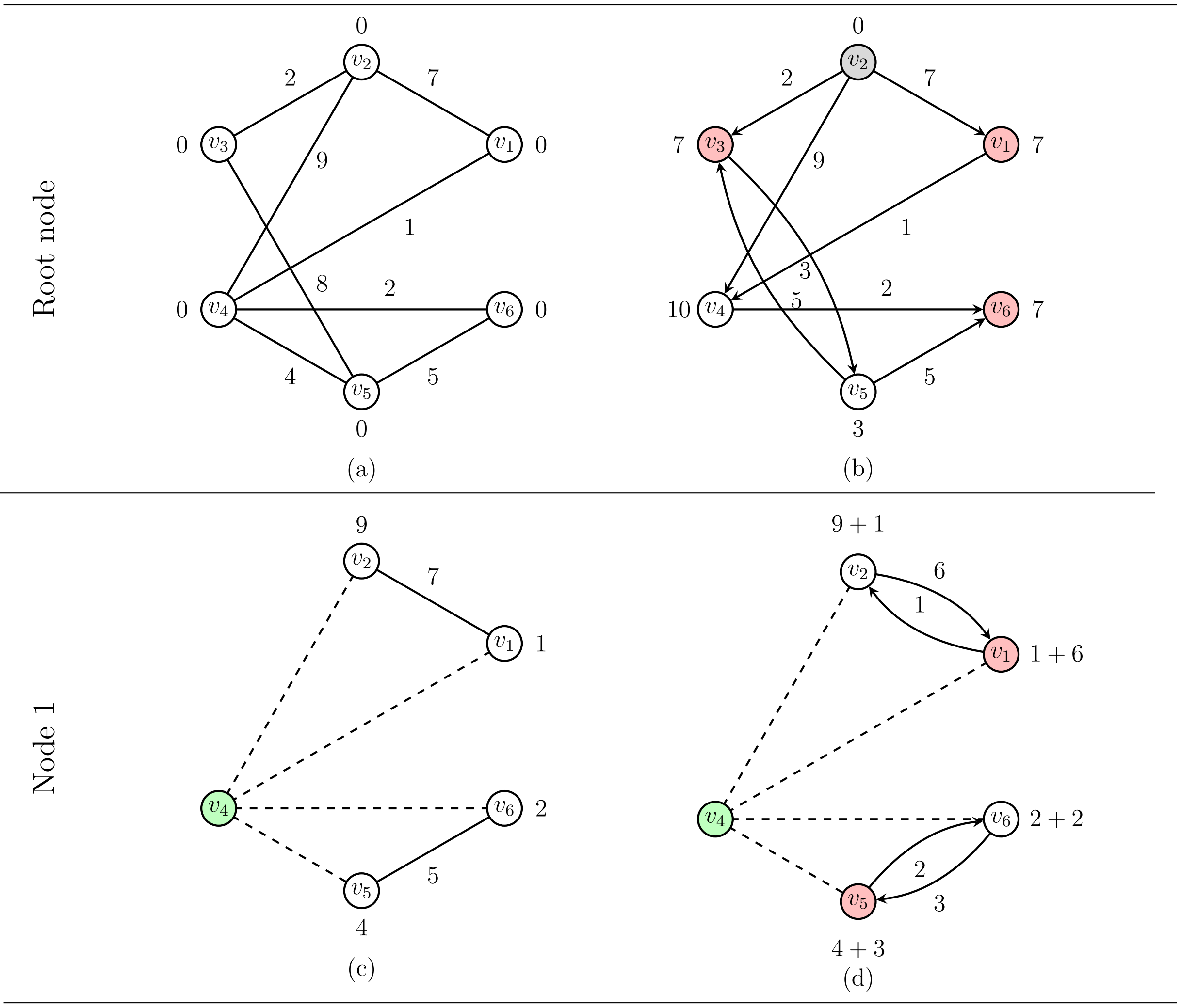

Figure 5: Illustration of the creation of the branching and pruned sets B and P for the root node (above) and node 1 (below) of the branch-and-bound tree of Figure 4. The subfigures parts (a) and (c) report $\hat{G}$ of the node with the corresponding vertex weights w_u. The subfigures parts (b) and (d) report the independent sets in $\tilde{\mathscr{I}}$ (in red and gray), the values of $\hat{w}_u$, those of $\tilde{\rho}^u_e$, and the partial clique $C = \{v_4\}$ (in green). For every $e = \{u, v\} \in \hat{E}$, an arc going from u to v represents the variable $\tilde{\rho}^v_e$ (only if strictly positive). The number associated with each vertex u is equal to $\tilde{w}_u$, written as the sum of w_u and $\sum_{e \in \delta(u,\hat{V})} \tilde{\rho}^u_e$.

which they are taken into account when branching, i.e., in this case v_4 is examined before v_5.

4.2. Construction of B and P for node 1

The set of vertices $\hat{V}$ is $\{v_1, v_2, v_5, v_6\}$, with $C = \{v_4\}$ and $c(C) = 0$. Note that all the vertices in $\hat{V}$ have positive weight.

The empty independent set I_1 is initialized (colored in red). The first vertex to be considered is v_1. Since $\texttt{budget}(v_1) = LB - c(C) - \tilde{w}_{v_1} = 7 - 0 - 0 - 1 = 6 \geq 0$, v_1 enters I_1. It absorbs 6 units from edge $\{v_1, v_2\}$. The remaining unit is pushed to v_2.

We then consider vertex v_5 (with $\tilde{w}(v_5) = 4$). Since $\texttt{budget}(v_5) = 7 - 0 - 0 - 4 = 3$, v_5 absorbs 3 units from edge $\{v_5, v_6\}$, pushing the remaining 2 to v_6.

Since no further vertices form an independent set with I_1, I_1 is closed. We let $\tilde{\pi}_{I_1} := \max\{7, 7\} = 7$.

No desorbtion takes place, as again the first and the second heaviest vertices in I_1 have the same weight (i.e., $\Delta = 0$).

A new empty independent set I_2 is generated. Since all the remaining vertices have negative budget (as $\tilde{\pi}_{I_1} = LB$ and their weights are all strictly positive), none is added to I_2 and I_2 is discarded.

BUILD_P terminates, producing $P = \{v_1, v_5\}$, with $UB(P) = 7$, and $B = \{v_2, v_6\}$.

5. Computational results

We present the results of an extensive set of experiments carried out to validate our new exact algorithm BBEWC for the MEWCP and to assess the impact of its features.

In the following, we compare the performance of BBEWC to five methods from the literature:

- CBQ: the exact algorithm for the MEWCP recently proposed in [14] and based on a quadratic programming formulation for the MEWCP (see Section 2).
- The following four compact ILP formulations, proposed in [11]:

F1: ILP formulation (4)–(9) in which the exponential family of Constraints (5) has been replaced by the following *nonedge* constraints:

$$x_u + x_v \leq 1, \qquad \{u, v\} \in \overline{E}. \tag{27}$$

F2: ILP formulation (4)–(9) in which the exponential family of Constraints (5) has been replaced by the following family:

$$\sum_{v \notin N(u)} x_v \leq (|V| - |\delta(u)| - 1)(1 - x_u), \qquad u \in V. \tag{28}$$

F11: Formulation F1 augmented with the following constraints, called *star inequalities* (see [29, 11]):

$$\sum_{e \in \delta(u)} y_e \leq (\omega(G) - 1)x_u, \qquad u \in V. \tag{29}$$

Differently from [11], which uses a heuristic upper bound on $\omega(G)$ to impose these inequalities, the exact value of $\omega(G)$ is used in our implementation, and hence, tighter cuts are generated. We calculate $\omega(G)$ using the algorithm proposed in [41] (which runs in a very short computing time for the instances in our test bed).

F21: Formulation F2 augmented with Constraints (29).

The four formulations are solved using the state-of-the art commercial MILP solver `CPLEX`, version 12.7.0 (called just `CPLEX` in what follows), in single-threaded mode, leaving all its parameters to their default value.

We refrain from carrying out a systematic comparison with the `EWCLIQUE` algorithm described in [42], as the authors' code is currently not publicly available and the results in the paper were carried out with a different machine. We will comment on the difference in performance between the two algorithms in Section 5.1.

All the experiments are carried out on a 20-core Intel(R) Xeon(R) CPU E5-2690 v2@3.00GHz, equipped with 128 GB of main memory and running a 64 bit Linux operating system. To facilitate the comparison with other codes run on a different machine, we report the performance of the clique algorithm `dfmax`[3], commonly used for calibration purposes, on the benchmark graphs r300.5, r400.5 and r500.5: the computing times are 0.189, 1.155, and 4.369 seconds, respectively (with a precision up to the millisecond).

Our algorithm is implemented in C++, compiled using gcc 4.8.4 (with optimization flag `-o3`), and run in a single-threaded environment on a single core. In all the experiments, the computing time is measured in seconds.

We consider four classes of instances, described in each of the following four subsections: classical DIMACS instances, weighted real-world networks, random Erdős-Rényi graphs, and real-world massive networks.

5.1. DIMACS instances

We consider a set of instances taken from the DIMACS 2 set [1], commonly used for testing maximum clique algorithms (see, e.g., [40, 39, 20, 8]). The edge weights are generated according to the following formula, as done in previous work on the MEWCP (see, e.g., [11, 14, 42]):

$$c_e = (u + v) \mod 200 + 1, \qquad e = \{u, v\} \in E. \tag{30}$$

We extract from the DIMACS benchmark a subset of graphs which are "tractable", in the sense that they can be solved in less than 7200 seconds by at least one of the algorithms we test. We obtain 36 graphs.

Table 1 reports the names of the instances, their size (in terms of the number of vertices $|V|$ and edges $|E|$), their clique number $\omega(G)$, and their edge-weighted clique number $\omega_c(G)$. Then, column (*#calls*) reports the number of recursive calls made by our algorithm `BBEWC`. The remaining columns show the computing times spent by each of the methods, i.e., `BBEWC`, `CBQ`, and the ILP formulations `F1`, `F2`, `F11`, and `F12` solved by `CPLEX`. The symbol *tl* indicates that the 7200 seconds time limit has been exceeded. The smallest computing time across the three algorithms is highlighted in boldface.

[3]http://archive.dimacs.rutgers.edu/pub/challenge/graph/solvers

Table 1: Results on the DIMACS instances.

					BBEWC		CBQ	F1	F11	F2	F12
	$\lvert V\rvert$	$\lvert E\rvert$	$\omega(G)$	$\omega_c(G)$	#calls	t. [s]	t. [s]	t. [s]	t. [s]	t. [s]	t. [s]
brock200_1	200	14,834	21	21,230	1,283,386	**24.32**	3531.06	t.l.	t.l.	t.l.	t.l.
brock200_2	200	9,876	12	6,542	3,252	**0.05**	8.47	6852.9	t.l.	3331.9	t.l.
brock200_3	200	12,048	15	10,303	30,696	**0.43**	64.53	t.l.	t.l.	t.l.	t.l.
brock200_4	200	13,089	17	13,967	70,848	**1.33**	212.59	t.l.	t.l.	t.l.	t.l.
C125.9	125	6,963	34	66,248	942,040	**25.82**	4938.78	6073.3	5468.2	t.l.	5475.7
c-fat200-1	200	1,534	12	7,734	13	**0.00**	0.60	137.8	60.6	38.7	25.2
c-fat200-2	200	3,235	24	26,389	4	**0.00**	0.96	232.8	219.4	45.6	31.4
c-fat200-5	200	8,473	58	168,200	12	**0.00**	t.l.	311.4	469.8	128.5	469.7
c-fat500-1	500	4,459	14	10,738	3	**0.00**	8.87	47.4	470.9	328.5	384.5
c-fat500-2	500	9,139	26	38,350	40	**0.00**	12.07	411.9	2065.1	526.6	2075.2
c-fat500-5	500	23,191	64	205,864	75	**0.01**	121.98	2968.8	3784.8	1054.5	3826.0
c-fat500-10	500	46,627	126	804,000	205	**0.06**	t.l.	4269.8	6544.2	4328.9	6588.0
dsjc500.5	500	62,624	13	9,626	514,010	**13.58**	2460.38	t.l.	t.l.	t.l.	t.l.
gen200_p0.9_44	200	17,910	44	94,362	67,608,313	**5804.55**	t.l.	t.l.	t.l.	t.l.	t.l.
gen200_p0.9_55	200	17,910	55	150,839	800,073	**180.06**	t.l.	t.l.	t.l.	t.l.	t.l.
hamming6-2	64	1,824	32	32,736	753	**0.02**	10.58	25.3	1.3	7.1	1.3
hamming6-4	64	704	4	396	29	**0.00**	0.07	15.4	1.3	7.2	1.6
hamming8-4	256	20,864	16	12,360	60,623	**1.41**	502.12	t.l.	3245.8	t.l.	3228.4
johnson8-2-4	28	210	4	192	9	**0.00**	0.01	1.3	0.0	0.1	0.0
johnson8-4-4	70	1,855	14	6,552	901	**0.01**	1.14	18.8	1.2	21.6	1.3
johnson16-2-4	120	5,460	8	3,808	427,100	**0.83**	99.90	286.8	8.0	3414.2	7.8
keller4	171	9,435	11	6,745	24,757	**0.27**	50.13	t.l.	4117.6	t.l.	4097.9
MANN_a9	45	918	16	5,460	58,432	**0.12**	2.70	46.6	1.3	38.2	1.3
p_hat300-1	300	10,933	8	3,321	452	**0.01**	3.38	1711.6	2839.0	1434.4	2837.4
p_hat300-2	300	21,928	25	31,564	28,092	**1.21**	198.99	t.l.	t.l.	t.l.	t.l.
p_hat500-1	500	31,569	9	4,764	5,559	**0.07**	25.67	t.l.	t.l.	t.l.	t.l.
p_hat700-1	700	60,999	11	5,185	18,527	**0.38**	105.92	t.l.	t.l.	t.l.	t.l.
p_hat1000-1	1000	122,253	10	5,436	66,430	**2.29**	489.40	t.l.	t.l.	t.l.	t.l.
p_hat1500-1	1500	371,746	12	7,135	628,108	**25.16**	4195.96	t.l.	t.l.	t.l.	t.l.
san200_0.7_1	200	13,930	30	45,295	12,141	**1.64**	t.l.	t.l.	1857.2	t.l.	1818.2
san200_0.7_2	200	13,930	18	15,073	33,683,211	**1376.22**	2515.19	t.l.	t.l.	t.l.	t.l.
san200_0.9_1	200	17,910	70	242,710	4,502	**2.11**	t.l.	t.l.	t.l.	t.l.	t.l.
san200_0.9_2	200	17,910	60	178,468	199,627	**61.17**	t.l.	5117.1	t.l.	t.l.	t.l.
san400_0.5_1	400	39,900	13	7,442	1,412,175	**129.50**	t.l.	t.l.	t.l.	t.l.	t.l.
sanr200_0.7	200	13,868	18	16,398	267,456	**4.85**	687.42	t.l.	t.l.	t.l.	t.l.
sanr400_0.5	400	39,984	13	8,298	198,228	**3.49**	593.33	t.l.	t.l.	t.l.	t.l.

Table 1 shows the clear superiority of BBEWC in terms of performance over the other algorithms. BBEWC is the fastest method over all the instances, achieving speedups of more than two orders of magnitude with respect to CBQ almost always. The difference in performance between BBEWC and the four ILP formulations solved with CPLEX is even larger. BBEWC manages to solve all the instances in well under the time limit of 7200, while CBQ fails to solve 8 instances and CPLEX with the best ILP formulation F12 fails to solve 18 instances.

The table also reveals that the san and gen families of instances are particularly difficult for both CPLEX and CBQ. CPLEX, in particular, fails to solve to optimality within the time limit most of these instances for all the ILP formulations. This also applies to the p_hat family for the case of CPLEX, but not CBQ. On the contrary, BBEWC performs very well for all the families: it solves 19 instances in under a second, 8 instances in less than 5 seconds, and all the remaining ones (with the sole exception of gen200_p0.9_44 and san200_0.7_2) in less than 200 seconds.

We report, in Figure 6, a performance profile which provides a graphical representation of the relative performance of the algorithms we have tested. For each instance, we compute the normalized time τ as the ratio of the computing time of the considered algorithm (which is ∞ if the instance is not solved to optimality) over the minimum computing time taken for all the algorithm we tested. For each value of τ on the horizontal axis, the vertical axis reports the percentage of instances for which the corresponding algorithm spent at most τ times the computing time of the fastest algorithm. At $\tau = 0$, the value of the curves is equal to the percentage of instances in which the corresponding algorithm is the fastest one. At the right-end of the chart (for the largest value of τ), each curve corresponds to the percentage of instances solved by a specific algorithm. The best performance is achieved by the algorithms whose curves occupy the upper part of the chart.

The performance profile confirms that BBEWC is very clearly the best algorithm for this set of instances, being the fastest one in 100% of the cases. The second best algorithm is CBQ, which solves more than 60% of the instances to proven optimality—the difference in performance between the two is, nevertheless, very large. The worst performance is achieved for the ILP formulations solved with CPLEX. The results for the F11 and F12 formulations (which include the star inequalities) are slightly better than those for the F1 and F2) formulations but, nevertheless, they are still substantially inferior to CBQ for every $\tau \geq 150$.

As mentioned, we cannot carry out a direct comparison with the results obtained for the combinatorial branch-and-bound algorithm EWCLIQUE described in [42], as we do not have access to the corresponding code. Nevertheless, as far as the DIMACS instances are concerned we can make the following observations. According to the results reported in [42], EWCLIQUE manages to solve 33 instances within a time limit of 1000 seconds. On the other hand, BBEWC solves 36 DIMACS instances within 7200 seconds, 34 of which under 1000 seconds. Moreover, on the six instances brock200_1, c-fat200-5, c-fat500-10, p_hat300-2, san200_0.7_1, and san200_0.9_2, BBEWC outperforms EWCLIQUE by more than one order of magnitude, whereas the opposite is true only for 2 instances (san200_0.7_2 and hamming8-2). Due to the differences in the machines used for the tests, a more precise and detailed comparison cannot be carried out.

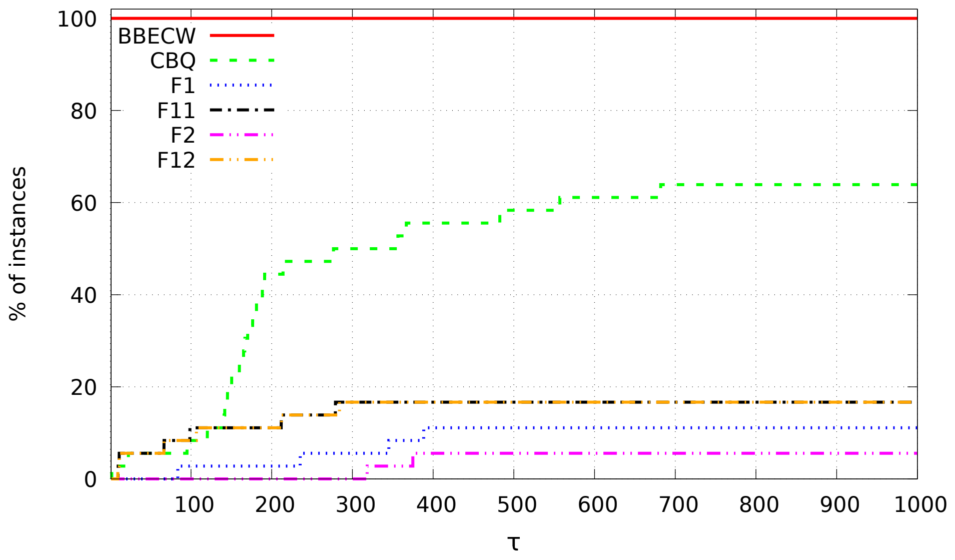

Figure 6: Performance profile on the DIMACS instances.

5.2. Weighted real-world networks

We report the results obtained on two sets of weighted real-world networks which have been frequently used in the recent literature for testing edge-weighted clique algorithms. Specifically, we consider a subset of the Reuters terror news (`RTN`) network proposed in [6] and the `SC-NIP` networks designed and tested in [11].

The instances from the `RTN` set are temporal networks based on the stories collected during 66 consecutive days by the Reuters news agency after the 09/11 terrorist attack (considered as day 0 in the instances). In this data set, each vertex represents a term (which appeared in the news) and two vertices are connected by an edge if and only if the two corresponding words appear in the same text unit. The weights on the edges count the number of times the corresponding couple of terms appeared in the same unit over the period of 66 days. In our tests, we consider a subset of 4 graphs as done in [14].[4] The graphs comprise the observations collected in days 1, 3, 7, and 15. They are denoted by `d1-RTN`, `d3-RTN`, `d7-RTN`, and `d15-RTN`, respectively. As done in [14], we preprocess the instances, which all have $|V| = 13308$ vertices, to remove their many isolated vertices. For additional information on these instances, we refer the interested reader to [11].

The `SC-NIP` instances correspond to two biological networks built from metabolic reaction information relative to the *saccharomyces cerevisiae* data (a type of yeast fungi) [9]. In the network `SC-NIP-m-t1`, the vertices are metabolites, with an edge between two metabolites if they share at least a reaction, i.e., if they both appear either as reactant or as the product of a chemical reaction. The weights correspond to the number of such reactions. In the second two networks `SC-NIP-r-t`k, for $k = 1, 2$, the vertices correspond to reactions and they are connected by an edge if and only if they share at least one metabolite. The edge weights

[4]These graphs were provided to us by the authors of [14].

correspond to the number of metabolites sharing the corresponding two reactions.

Table 2 shows the results obtained on the seven graphs in this data set.

Table 2: Results on the weighted real-world networks.

					BBEWC		CBQ	F1	F11	F2	F12
	$\|V\|$	$\|E\|$	$\omega(G)$	$\omega_c(G)$	#calls	t. [s]	t. [s]	t. [s]	t. [s]	t. [s]	t. [s]
d1-RTN	1476	7,734	10	4,524	3	**0.00**	15.47	84.6	673.4	520.6	591.6
d3-RTN	3125	24,145	18	5,859	11	**0.01**	177.09	1292.1	t.l.	t.l.	t.l.
d7-RTN	4420	41,009	18	7,424	23	**0.01**	519.34	2887.2	t.l.	t.l.	t.l.
d15-RTN	5498	57,845	18	7,424	98	**0.02**	976.48	t.l.	t.l.	t.l.	t.l.
SC-NIP-m-t1	770	3,825	9	343	33	**0.00**	2.39	22.9	104.0	114.7	129.2
SC-NIP-r-t1	1349	56,218	174	25,290	36	**0.02**	752.83	251.9	972.1	1509.7	1723.6
SC-NIP-r-t2	739	17,479	121	15,188	1	**0.00**	19.94	109.5	171.0	110.9	147.5

The table provides the same information as that in Table 1, i.e., the number of vertices $|V|$, the number of edges $|E|$, the clique number $\omega(G)$, the edge-weighted clique number $\omega_c(G)$, the number of calls to the BBEWC algorithm, and the computing time for each of the tested algorithms. The results show that BBEWC can solve all the instances in a fraction of a second, while CBQ and the four ILP formulations are slower by several orders of magnitude in all cases. CPLEX exceeds the time limit of 7200 seconds for all four ILP formulations on the instance d15-RTN, and F1 is the only formulation which solves to optimality the instances d3-RTN and d7-RTN within the time limit. Finally, we note that CBQ outperforms CPLEX in all cases except for the graph SC-NIP-r-t1, where it is around three times slower than F1.

Overall, the table shows that BBEWC substantially outperforms all the other methods also on this data set. We note that the performance of BBEWC remains unchanged even when run on the *raw* RTN graphs (rather than on their preprocessed counterparts). This is not the case for the other approaches we tested, for which we observe a significantly worse performance when the preprocessing phase is neglected.

5.3. Random instances

We consider a data set encompassing 180 Erdős-Rényi random graphs $G(n, \mu)$ of different sizes $|V|$, ranging from 150 to 10000, and different edge densities (see Table 3 for the edge density value μ of each (n, μ) family of instances). By construction, for each pair of vertices in $u, v \in V$ these graphs contain an edge $e = \{v, u\}$ with a uniform probability equal to the edge density μ. Similar graphs are very often used for testing max clique algorithms, see, e.g., [41]. We consider 10 random instances for each (n, μ) combination of size and density reported in the table. The edge weights are generated as for the DIMACS instances according to Formula (30).

For each (n, μ) set of 10 graphs, Table 3 also reports the range of the clique number $\omega(G)$ and the average maximum edge-weighted clique number $\omega_c(G)$ computed by our BBEWC algorithm. Finally, the last two columns of the table show the average number of recursive calls (*#calls*)

Table 3: Results on random Erdős-Rényi instances.

				BBEWC	
$\|V\|$	μ	$\omega(G)$	$\omega_c(G)$	#calls	t. [s]
150	0.7	16-17	14,739.6	41,853	0.5
150	0.8	23	28,556.2	431,443	8.1
150	0.9	35-38	173,506.6	12,895,115	232.9
200	0.7	17-18	16,737.9	329,136	5.5
200	0.8	25-26	31,692.0	8,861,123	222.1
300	0.6	15-16	12,689.2	331,924	6.7
300	0.7	20-21	20,802.8	7,992,047	188.9
500	0.4	10-11	5,838.0	56,905	1.2
500	0.5	13	9,416.7	594,690	14.4
500	0.6	17	15,347.1	13,258,071	370.2
1000	0.2	7-8	3,095.2	9,204	0.2
1000	0.3	9-10	4,831.0	96,870	3.7
1000	0.4	12	7,695.0	2,091,692	64.2
3000	0.1	6-7	2,402.5	3,780	1.1
3000	0.2	9	4,222.7	484,107	43.7
5000	0.1	7	2,668.8	43,941	7.5
5000	0.2	9-10	4,862.4	2,592,958	522.0
10000	0.1	7-8	3,234.1	1,184,516	156.9

and the average computing time required by BBEWC to solve them to optimality. As can be seen, BBEWC solves all the instances in under 600 seconds.

The table shows that, given a value of $|V|$, the instances get harder to solve for increasing densities, a typical phenomenon in max clique problems, and that the speed at which their computing time grows is faster for larger values of $|V|$. For instance, it only takes 188.9 seconds to solve, on average, instances with $|V| = 300$ and a density $\mu = 0.7$, whereas, with $|V| = 10000$, an average of 156.9 seconds are necessary already for $\mu = 0.1$.

It is interesting to point out that the the MEWCP is notably harder to solve than the MCP, at least in this data set. This can be seen by comparing the results we reported with those in [39], which employs a state-of-the-art MCP solver for the MCP running on the same hardware. For example, the average computing times reported for the MCP algorithm NEW in [41] (Table 2) for instances with $|V| = 300$ and a density $\mu = 0.7$ is of 2.5 seconds (whereas BBEWC requires 188.9 seconds), and 25.3 seconds for the instances with $|V| = 10000$ and a density of $\mu = 0.1$ (whereas BBEWC requires 156.9 seconds).

5.4. Massive real-world networks

Finally, we analyse the behavior of our algorithm on massive real-world networks. We consider a set of instances with clique number $\omega(G) \geq 10$ taken from the SNAP database [18]. They belong to the following three categories:

- `Social Networks` are, generally speaking, graphs whose vertices represent individuals and an edge indicates that two individuals share some kind of relationship. Interaction networks encompass the cases where the edges indicate an interaction via the exchange of a message (e.g., in `ia-email-EU` the message is an email). In recommendation networks, an edge indicates the exchange of a recommendation or of an opinion between individuals (e.g., `rec-eachmovie` indicates the exchange of a movie preference). In collaboration networks, the edges identify a work-related collaboration (e.g., in `astro-ph` and `cond-mat` the relationship is between two researchers who coauthored a preprint uploaded on the "astro physics" or "condensed matter" segment of arXiv). The relationship is of "friendship" on Facebook in `socfb`, and of interaction between bloggers in `soc-BlogCatalog`.
- `Technological networks` are networks whose vertices represent routers and whose edges indicate a communication between a pair of them, as in `tech-internet-as`.
- `Scientific computing networks` are mesh graphs derived from mathematical analyses carried out in different disciplines, such as, e.g., finite elements (`sc-` instances).

The weights of these instances are constructed according to Equation (30).

For each instance, Table 4 reports its size in terms of number of vertices $|V|$ and edges $|E|$, its clique number $\omega(G)$, its edge-weighted clique number $\omega_c(G)$ computed with `BBEWC`, the number of recursive calls performed by our algorithm, and the computing time in seconds. As the table shows, `BBEWC` manages to solve all the instances to optimality in less than 30 minutes. Moreover, with the sole exception of 4 cases all the instances are solved in well under a minute. One can see that, in general, larger computing times go hand in hand with a high density. This applies to the harder instances, `socfb-UIllinios`, `socfb-UF`, `socfb-Texas84`, and `soc-themaker`.

We would like to highlight that, to the best of our knowledge, this is the first time that these real-world instances have been solved to optimality for the MEWCP. We believe this to be a further practical contribution of our work, as it envisages the possibility of employing the edge-weighted clique number as one of the characteristic numbers for social networks endowed with edge-weighted information quantifying the level of (social) interaction between their vertices (as it is the case for the sparse real-world networks in Section 5.2.)

5.5. On the impact of the DESORB phase and of our bounding procedure

We analyse, in this section, the effect of the DESORB phase on the overall performance of the algorithm, also illustrating the pruning power of the bounding procedure proposed in this work. Table 5 illustrates on a subset of 9 DIMACS instances the total number of nodes (`# total`) crated by `BBEWC`, the number of nodes which were pruned (`# pruned`) due to being part of the pruned set P, and their percentage (%) as a fraction of $\hat{V}$ without (left) and with (right) carrying out the DESORB phase. We remark that the number of nodes that we report corresponds to the number of subproblems examined by the algorithm, which is not the same as the number of recursive calls—for example, in the demonstration example described in Figure 4 the total number of nodes examined by `BBEWC` is 7, whereas the number of calls is just 4.

Table 4: Results on the massive real-world instances taken from the SNAP repository.

					BBEWC	
	$\lvert V\rvert$	$\lvert E\rvert$	$\omega(G)$	$\omega_c(G)$	#calls	t. [s]
socfb-UIllinois	30,795	1,264,421	57	160,092	210,611	73.94
cond-mat-2003	31,163	120,029	25	42,324	227	0.32
ia-email-EU	32,430	54,397	12	8,155	192	0.08
rgg_n_2_15_s0	32,768	160,240	13	8,058	50	0.40
ia-enron-large	33,696	180,811	20	19,860	1,816	0.64
socfb-UF	35,111	1,465,654	55	149,419	1,417,532	623.44
socfb-Texas84	36,364	1,590,651	51	129,925	838,917	394.95
tech-internet-as	40,164	85,123	16	13,695	168	0.19
cond-mat-2005	40,421	175,691	30	42,324	178	0.55
sc-nasasrb	54,870	1,311,227	24	51,040	260	3.60
soc-brightkite	56,739	212,945	37	67,102	29,550	6.29
soc-loc-brightkite	58,228	214,078	37	79,678	49,430	9.42
rgg_n_2_16_s0	65,536	342,127	14	9,711	121	1.62
soc-themarker	69,413	1,644,843	22	23,605	1,136,712	1104.91
rec-eachmovie	74,424	1,634,743	12	7,791	22,930	37.11
soc-Slashdot0811	77,360	469,180	26	35,450	13,054	6.14
soc-Slashdot0902	82,168	504,230	27	34,951	17,082	9.76
sc-pkustk11	87,804	2,565,054	36	77,580	1,179	10.41
ia-wiki-Talk	92,117	360,767	15	12,422	3,898	3.93
sc-pkustk13	94,893	3,260,967	36	99,915	6,719	17.71

Table 5: Impact of the DESORB phase on a subset of DIMACS instances.

	BBEWC (no desorbing)				BBEWC (with desorbing)			
		nodes				nodes		
	t. [s]	# total	# pruned	%	t. [s]	# total	# pruned	%
brock200_1	32.5	11,973,064	10,404,958	86.9	24.3	8,071,664	6,788,278	84.1
C125.9	54.1	9,670,404	8,247,173	85.3	25.8	3,712,162	2,770,123	74.6
dsjc500.5	15.4	7,359,624	6,735,639	91.5	13.6	5,706,659	5,192,650	91.0
gen200_p0.9_44	9442.8	778,167,572	688,443,365	88.5	5804.5	429,401,459	361,793,146	84.3
gen200_p0.9_55	277.2	14,634,737	13,655,871	93.3	180.1	8,339,910	7,539,837	90.4
p_hat1500-1	25.5	9,187,573	8,317,942	90.5	25.2	7,124,764	6,496,646	91.2
san200_0.7_2	1791.8	384,165,586	337,211,091	87.8	1376.2	274,328,570	240,645,344	87.7
san200_0.9_2	102.3	4,212,944	3,967,509	94.2	61.2	2,396,855	2,197,229	91.7
san400_0.5_1	135.5	28,516,995	26,950,815	94.5	129.5	25,880,541	24,468,354	94.5

Averaging the data of Table 5, we notice that the impact of the DESORB phase is quite substantial, as it allows for a reduction in the total computing time by a factor of 1.6 (reduced from 11,877.1 seconds to 7,640 seconds) and in the total number of nodes by a similar factor (from 1.24 billions to 0.76 billions).

From Table 5 it also emerges that, regardless of the DESORB phase being used or not, the percentage of nodes that are pruned is very significant—equal to 90.3% when DESORB is applied and to 87.7% when it is not. This reveals the effectiveness of the bounding procedure that we have proposed for the problem.

6. Conclusions

We have proposed a new combinatorial branch-and-bound algorithm for the MEWCP based on a novel bounding procedure with a high pruning capability. The extensive computational results that we have provided (on random and structured graphs, encompassing standard benchmarks used in the literature as well as recently introduced real-world large-scale graphs) show that our new branch-and-bound algorithm improves on previous state-of-the-art exact approaches from the literature by up to several orders of magnitude.

Interesting and challenging directions for future work include the parallelization of our algorithm and the development of improved bounding techniques. It would also be of interest to extend our algorithm to other problems related to the MEWCP, such as the *maximum diversity*, *k-subgraph*, or *k-cluster problems*. We also believe that the computational framework that we have proposed offers a good starting point for addressing edge-weighted clique problems with high centrality measures, whose interest has been recently shown in social network analysis [33]. Finally, the study of edge-weighted clique problems under uncertainty and in a bilevel setting would certainly be a further promising area of research (see, e.g., [32, 10, 5] for related works on vertex-weighted problems of this type).

7. Acknowledgments

This work is partially funded by the Spanish Ministry of Science, Innovation and Universities through the project COGDRIVE(grant DPI2017-86915-C3-3-R). The authors would also like to thank S. Hosseinian for sharing the source code of the algorithm `CBQ` and for his collaboration when comparing our algorithm with `CBQ`.

References

[1] 2nd DIMACS Implementation Challenge - NP Hard Problems: Maximum Clique, Graph Coloring, and Satisfiability, 2017. URL `http://dimacs.rutgers.edu/Challenges/`.

[2] L. A. Agapito, M. Fornari, D. Ceresoli, A. Ferretti, S. Curtarolo, and M. B. Nardelli. Accurate tight-binding hamiltonians for two-dimensional and layered materials. *Phys. Rev. B*, 93:125–137, 2016.

[3] J. Brown, K. D. Bahadur, E. Tomita, and T. Akutsu. Multiple methods for protein side chain packing using maximum weight cliques. *Genome Informatics*, 17(1):3–12, 2006.

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

[5] S. Coniglio and S. Gualandi. On the separation of topology-free rank inequalities for the max stable set problem. In *LIPIcs-Leibniz International Proceedings in Informatics*, volume 75. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

[6] S. Corman, T. Kuhn, R. Mcphee, and K. Dooley. Studying complex discursive systems centering resonance analysis of communication. *Human Communication Research*, 28(2): 157–206, 4 2002.

[7] G. Dijkhuizen and U. Faigle. A cutting-plane approach to the edge-weighted maximal clique problem. *European Journal of Operational Research*, 69(1):121 – 130, 1993.

[8] Z. Fang, C. Li, and K. Xu. An exact algorithm based on maxsat reasoning for the maximum weight clique problem. *J. Artif. Intell. Res.*, 55:799–833, 2016.

[9] J. Forster, I. Famili, P. Fu, B. Palsson, and J. Nielsen. Genome-scale reconstruction of the Saccharomyces cerevisiae metabolic network. *Genome Res.*, 13(2):244–253, 2003.

[10] F. Furini, I. Ljubić, P. S. Segundo, and S. Martin. The maximum clique interdiction problem. 2018. under revision.

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

[12] M. Grötschel, L. Lovász, and A. Schrijver. The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica*, 1(2):169–197, 1981.

[13] S. Hosseinian, D. B. M. M. Fontes, S. Butenko, M. B. Nardelli, M. Fornari, and S. Curtarolo. *The Maximum Edge Weight Clique Problem: Formulations and Solution Approaches*, pages 217–237. Springer International Publishing, 2017.

[14] S. Hosseinian, D. B. M. M. Fontes, and S. Butenko. A nonconvex quadratic optimization approach to the maximum edge weight clique problem. *J. Global Optimization*, 72(2): 219–240, 2018.

[15] M. Hunting, U. Faigle, and W. Kern. A Lagrangian relaxation approach to the edge-weighted clique problem. *European Journal of Operational Research*, 131(1):119–131, 2001.

[16] H. Jiang, C. Li, and F. Manyà. An exact algorithm for the maximum weight clique problem in large graphs. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, pages 830–838, 2017.

[17] T. L. Lei and R. L. Church. On the unified dispersion problem: Efficient formulations and exact algorithms. *European Journal of Operational Research*, 241(3):622–630, 2015.

[18] J. Leskovec and A. Krevl. SNAP Datasets: Stanford large network dataset collection, June 2014. URL `http://snap.stanford.edu/data`.

[19] C. Li, Z. Fang, and K. Xu. Combining MaxSAT reasoning and incremental upper bound for the maximum clique problem. In *2013 IEEE 25th International Conference on Tools with Artificial Intelligence*, pages 939–946, 2013.

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

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

[22] C. Li, Y. Liu, H. Jiang, F. Manyà, and Y. Li. A new upper bound for the maximum weight clique problem. *European Journal of Operational Research*, 270(1):66–77, 2018.

[23] X. Li, M. Wu, C.-K. Kwoh, and S.-K. Ng. Computational approaches for detecting protein complexes from protein interaction networks: a survey. *BMC genomics*, 11(1):S3, 2010.

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

[25] E. M. Macambira and C. C. de Souza. The edge-weighted clique problem: Valid inequalities, facets and polyhedral computations. *European Journal of Operational Research*, 123(2):346–371, 2000.

[26] R. Martí, M. Gallego, A. Duarte, and E. G. Pardo. Heuristics and metaheuristics for the maximum diversity problem. *Journal of Heuristics*, 19(4):591–615, 2013.

[27] T. S. Motzkin and E. G. Straus. Maxima for graphs and a new proof of a theorem of Turán. *Canad. J. Math*, 17(4):533–540, 1965.

[28] P. R. J. Östergård. A fast algorithm for the maximum clique problem. *Discrete Appl. Math.*, 120(1-3):197–207, 2002.

[29] K. Park, K. Lee, and S. Park. An extended formulation approach to the edge-weighted maximal clique problem. *European Journal of Operational Research*, 95(3):671 – 682, 1996.

[30] O. A. Prokopyev, N. Kong, and D. L. Martinez-Torres. The equitable dispersion problem. *European Journal of Operational Research*, 197(1):59–67, 2009.

[31] V. Rutenburg. Propositional truth maintenance systems: Classification and complexity analysis. *Annals of Mathematics and Artificial Intelligence*, 10(3):207–231, 1994.

[32] M. Rysz, M. Mirghorbani, P. A. Krokhmal, and E. L. Pasiliao. On risk-averse maximum weighted subgraph problems. *J. Comb. Optim.*, 28(1):167–185, 2014.

[33] M. Rysz, F. M. Pajouh, and E. L. Pasiliao. Finding clique clusters with the highest betweenness centrality. *European Journal of Operational Research*, 271(1):155–164, 2018.

[34] P. San Segundo and J. Artieda. A novel clique formulation for the visual feature matching problem. *Appl. Intell.*, 43(2):325–342, 2015.

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

[36] P. San Segundo and C. Tapia. Relaxed approximate coloring in exact maximum clique search. *Computers & OR*, 44:185–192, 2014.

[37] P. San Segundo, D. Rodriguez-Losada, and A. Jimenez. An exact bit-parallel algorithm for the maximum clique problem. *Computers & OR*, 38(2):571–581, 2011.

[38] P. San Segundo, F. Matia, D. Rodriguez-Losada, and M. Hernando. An improved bit parallel exact maximum clique algorithm. *Optimization Letters*, 7(3):467–479, 2013.

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

[40] P. San Segundo, A. Lopez, M. Batsyn, A. Nikolaev, and P. M. Pardalos. Improved initial vertex ordering for exact maximum clique search. *Appl. Intell.*, 45(3):868–880, 2016.

[41] P. San Segundo, A. Nikolaev, M. Batsyn, and P. M. Pardalos. Improved infra-chromatic bound for exact maximum clique search. *Informatica, Lith. Acad. Sci.*, 27(2):463–487, 2016.

[42] S. Shimizu, K. Yamaguchi, and S. Masuda. A branch-and-bound based exact algorithm for the maximum edge-weight clique problem. In *Computational Science/Intelligence & Applied Informatics, CSII2018*, pages 27–47, 2018.

[43] M. M. Sørensen. New facets and a branch-and-cut algorithm for the weighted clique problem. *European Journal of Operational Research*, 154(1):57–70, 2004.

[44] E. Tomita, T. Akutsu, and T. Matsunaga. Efficient algorithms for finding maximum and maximal cliques: Effective tools for bioinformatics. In *Biomedical engineering, trends in electronics, communications and software*. InTech, 2011.

[45] Q. Wu and J. Hao. An adaptive multistart tabu search approach to solve the maximum clique problem. *J. Comb. Optim.*, 26(1):86–108, 2013.