Models and Algorithms for the Weighted Safe Set Problem

Enrico Malaguti[1]

Dipartimento di Ingegneria dell'Energia Elettrica e dell'Informazione "Guglielmo Marconi"
Università di Bologna
Bologna, Italy

Vagner Pedrotti

Faculty of Computing
Federal University of Mato Grosso do Sul
Campo Grande, Brazil

Abstract

Given a connected graph $G = (V, E)$, a *Safe Set* S is a subset of the vertex set V such that the cardinality of each connected component in the subgraph induced by $V \setminus S$ does not exceed the cardinality of any neighbor connected component in the subgraph induced by S. When the vertices of G are weighted, the weight of a component is defined as the sum of the weights of its vertices, and the notion of safe set is extended by considering the weight of connected components in subgraphs induced by S and by $V \setminus S$. We propose an integer linear formulation which can tackle the four variants of the problem which arise by imposing connectivity of the safe set, and by considering weighted or unweighted vertices, respectively. Despite alternative formulations from the literature, that require a large number of variables, our formulation only uses one variable per vertex. The formulation has an exponential number of constraints, which are needed to define the structure of the safe set, and can be generated on-the-fly within a branch-and-cut algorithm. We describe linear-time separation procedures for these constraints, as well as families of additional inequalities based on cliques and on minimum weight cut separators, and discuss separation algorithms. A branch-and-cut algorithm that solves the proposed formulation is computationally compared with the state-of-the-art alternative formulation from the literature, and shows faster in solving most of benchmark instances.

Keywords: Safe Set Problem, Weighted Safe Set Problem, Branch-and-cut Algorithm, Computational experiments.

Email addresses: enrico.malaguti@unibo.it (Enrico Malaguti), vagner.pedrotti@ufms.br (Vagner Pedrotti)

[1]Supported by the Air Force Office of Scientific Research under award FA8655-20-1-7019

1. Introduction

Given an undirected connected graph $G = (V, E)$, where V is the vertex set and E is the edge set, a *Safe Set* is a subset S of the vertex set V having cardinality at least equal to the cardinality of any other connected component in the graph. By considering nonnegative weights associated with the vertices of G, a *Weighted Safe Set* has at least the same weight of any other connected component in the graph. The *Safe Set Problem* and the *Weighted Safe Set Problem* ask to find a minimum cardinality safe set and a minimum weight safe set of a graph, respectively. In the *connected* versions of those problems, the set S is required to induce a connected subgraph of G.

The Safe Set Problem has some relevant applications: in network design, a Safe Set represents a safe location, having enough capacity (weight) to give shelter to neighbor nodes [1]; in network analysis, a Safe Set is interpreted as a minimum weight leading subnetwork in a larger network [2].

The Safe Set Problem was originally introduced in [1], together with its connected variant. Polynomial time algorithms for the Safe Set Problem on trees, interval graphs and graphs with bounded treewidth were described in[3]. The authors also give a fixed-parameter tractable algorithm using the solution size as parameter. An analysis of the parameterized complexity of the problem is also discussed in [4]. The paper presents hardness results and fixed-parameter algorithms. The Weighted Safe Set Problem was firstly introduced in [2], where NP-Hardness for trees and even stars is proved.

The Safe Set Problem and Weighted Safe Set Problem have recently attracted attention on the computational side as well. In Macambira et. al [5], the authors proposed an Mixed Integer Linear Programming (MILP) formulation requiring an exponential number of constraints. Variables of this model describe each connected component of the safe set, as well as the connected components of the graph induced by the remaining vertices. A representative vertex identifies each component, which is denoted explicitly by the variable index. Overall this formulation defines $O(|V|^3)$ variables, and an exponential number of constraints is necessary to impose connectivity of the components. The model is tackled by a branch-and-cut algorithm. The paper reports a set of computational experiments performed on pseudo-random instances.

Hosteins [6] presented a compact MILP formulation based on network flows. Also in this model variables are used to enumerate connected components and their vertices either in the graph induced by the safe set, and in the graph induced by the remaining vertices. This requires $O(|V|^2)$ variables and a polynomial number of constraints. The model is tested on the set of randomly generated instances, as well as on the instances proposed in [5].

Both formulations [5, 6] exploit variables to give an explicit description of the connected components in which graph $G[V \setminus S]$ disconnects. The number of needed variables can grow considerably for large graphs, because the number of components in an optimal solution can be large. This is the case in particular when the considered graph is sparse. Our approach is different in the sense that we keep the number of variables of our formulation as small as possible, indeed,

we only define indicator variables for set S. We then impose feasibility of the defined set S through an exponential family of constraints, that we generate on-the-fly within a branch-and-cut algorithm, while minimizing the cardinality or weight of S. The aim is to use a lighter model whose linear relaxation can be solved faster, in the spirit of the approach followed by Fischetti et. al [7] for the Steiner Tree Problem, and more recently by Furini, Ljubić, Malaguti and Paronuzzi [8], [9] for Vertex Separator problems.

We conclude this section by introducing some notation and by giving a formal definition of the problems at study. For a subset S of V and any vertex $v \in V$, we denote by $G[S]$ the subgraph of G induced by S, and we define P_v as the connected component of $G[S]$ which contains v, if $v \in S$; or as the connected component of $G[V \setminus S]$ which contains v, otherwise. We also denote the neighbors of S as $N(S) = \{u \in V \setminus S : \exists v \in S, uv \in E\}$. A set S is a *safe set* of G if for every edge uv of G such that $u \in S$ and $v \notin S$, $|P_u| \geq |P_v|$. Such components P_u and P_v are called neighbor components. Given a weight function $w : V \to \mathbb{N}^+$ to assign to each vertex a non-negative integer, let the weight of a subset X of V, denoted $w(X)$, be the sum of the $w(v)$ for every $v \in X$. Then, the set S is a *weighted safe set* of G with respect to w if for every edge uv of G such that $u \in S$ and $v \notin S$, $w(P_u) \geq w(P_v)$.

The remainder of the paper is organized as follows: in Section 2 we present the new formulation for the Weighted Safe Set Problem. Additional strengthening inequalities are presented in Section 2.2. The branch-and-cut algorithm and separation procedures are discussed in Section 3. Finally, we report computational experiments in Section 4, and we draw some conclusions in Section 5.

2. ILP formulation

Let C and C' be disjoint subsets of V. We denote by $\mathcal{P}$ the collection of such pairs (C, C') for which: $w(C) < w(C')$; $G[C]$ and $G[C']$ are connected subgraphs; and there exists an edge uv in G with $u \in C$ and $v \in C'$.

Our formulation for the Weighted Safe Set Problem is based on the observation that for any pair $(C, C') \in \mathcal{P}$ and any safe set S of G having nonempty intersection with C, either $S \cap N(C) \neq \emptyset$ or $S \cap C' \neq \emptyset$. We first prove this statement in the following Lemma 1, and then show, in Lemma 2, that this condition is also sufficient for S to be a safe set.

Lemma 1. *Let S be a safe set and $(C, C') \in \mathcal{P}$ such that $S \cap C \neq \emptyset$. Then either $S \cap C' \neq \emptyset$ or $S \cap N(C) \neq \emptyset$.*

Proof. We will show that if S is a safe set of G and $G[C]$,$G[C']$ are connected subgraphs of G, and there are $v \in C$ and $u \in C'$ such that $uv \in E(G)$, $S \cap C \neq \emptyset$, and $S \cap (C' \cup N(C)) = \emptyset$, then $w(C) \geq w(C')$, which implies $(C, C') \notin \mathcal{P}$. This is enough for a proof by contradiction.

Let $a \in C \cap S$ and let $A \subset V$ such that $G[A]$ is the maximal connected component of $G[S]$ such that $a \in A$. Since $S \cap N(C) = \emptyset$, $A \subseteq C$, and hence we have $w(A) \leq w(C')$. Next, consider the subset $B \subset V$, such that $G[B]$ is the

maximal connected component of $G[V \setminus S]$ that contains u. Moreover, because $C' \subset V \setminus S$ and C' is connected, $C' \subseteq B$, and hence we have $w(B) \geq w(C')$.

Since $G[C]$ and $G[C']$ are connected, and $uv \in E(G)$, there is an edge in G connecting a vertex in B to a vertex in A. Since S is a safe set, $w(A) \geq w(B)$. From the previous observations, we have $w(C) \geq w(A) \geq w(B) \geq w(C')$, which completes the proof. □

Lemma 2. *Let $X \subset V$ and $X \neq \emptyset$. If X is not a safe set of G, there is a pair $(C, C') \in \mathcal{P}$ such that $X \cap C \neq \emptyset$ but $X \cap C' = \emptyset$ and $X \cap N(C) = \emptyset$.*

Proof. Since $X \neq \emptyset$ and X is not a safe set of G, there exist sets $A \subset V$ and $B \subset V$ such that $G[A]$ is a maximal connected subgraph of $G[X]$ and $G[B]$ is a maximal connected subgraph of $G[V \setminus X]$, and there are $v \in A$ and $u \in B$ with $uv \in E(G)$ and $w(A) < w(B)$. Thus, $(A, B) \in \mathcal{P}$ while $X \cap A \neq \emptyset$ and $X \cap B = \emptyset$ by construction. Finally, $X \cap N(A) = \emptyset$, otherwise $G[A]$ is not a maximal connected subgraph of $G[X]$. □

Our MILP formulation for the Weighted Safe Set Problem only uses indicator variables for set S. We define one binary variable x_v for each vertex $v \in V$, taking value 1 if vertex v belongs to the safe set S, and 0 otherwise. When $w(v) = 1$ for every $v \in V$, the formulation models the (unweighted) Safe Set Problem. The formulation reads

$$\min \sum_{v \in V} w(v) x_v \tag{1}$$

$$\text{s.t.} \sum_{v \in V} x_v \geq 1 \tag{2}$$

$$\sum_{v \in C' \cup N(C)} x_v \geq x_l \qquad (C, C') \in \mathcal{P}, l \in C \tag{3}$$

$$x_v \in \{0, 1\} \qquad v \in V. \tag{4}$$

Inequality (2) enforces that at least one vertex is selected and inequalities (3) ensure the solution satisfies the conditions of Lemma 2 which are sufficient to define a safe set. We denote this last exponential-size collection of inequalities as Neighbor Components inequalities.

2.1. Connected Safe Set

Formulation (1)–(4) could define a disconnected safe set S. In case it is required, connectivity of S can be obtained by following the same idea exploited in [7] to impose connectivity of Steiner trees. Given a pair of vertices $u, v \in G$, let $\mathcal{A}(i, j)$ be the collection of all $(i, j)-$separators. If $u, v \in S$, than at least one vertex from any separator $A \in \mathcal{A}(i, j)$ has to be in S as well, as specified by the following *connectivity constraints*

$$\sum_{l \in A} x_l \geq x_u + x_v - 1 \qquad u, v \in V,\ u \neq v,\ A \in \mathcal{A}(i, j). \tag{5}$$

2.2. Strengthening inequalities

Despite formulation (1)-(4) is sufficient to define the largest weight safe set of a graph, its linear relaxation can be strengthened so as to reduce its integrity gap. In this section we present some inequalities designed with this scope.

2.2.1. Clique and q-connected Components

Let us consider a connected graph $G = (V, E)$, a clique $K \subseteq V$ and a subset of the vertex set $S \subseteq V$. We have that $G[K \setminus S]$ is a connected subgraph and there is a path in G connecting each vertex in $K \setminus S$ to some vertex in S. Hence, in order for S to be a safe set, it must be

$$w(S) \geq w(K \setminus S),$$

which can be rewritten as

$$\begin{aligned} w(K \cap S) + w(S \setminus K) &\geq w(K) - w(K \cap S) \\ 2w(K \cap S) + w(S \setminus K) &\geq w(K). \end{aligned}$$

Hence, a vector of binary variables associated with the vertices $x_v,\ v \in V$ representing a weighted safe set must satisfy the following inequality

$$2 \sum_{v \in K} w(v)x_v + \sum_{v \in V \setminus K} w(v)x_v \geq w(K). \tag{6}$$

We can generalize inequality (6) by considering a subset $Q \subset V$ of vertices inducing a connected subgraph $G[Q]$ instead of a clique. Let $w_q(Q)$ be the minimum weight of a subset of vertices whose removal from Q would disconnect the subgraph. The value $w_q(Q)$ can be computed by solving a max-flow problem for each pair of vertices in Q [10].

If S is a safe set, either $G[Q \setminus S]$ is connected and $w(S) \geq w(Q \setminus S)$, or $G[Q \setminus S]$ is disconnected which happens when $w(S) \geq w_q(Q)$. We express these conditions as

$$\begin{aligned} w(S) &\geq \min\{w(Q \setminus S), w_q(Q)\} \\ &= \min\{w(Q) - w(Q \cap S), w_q(Q)\} \\ &= w(Q) - \max\{w(Q \cap S), w(Q) - w_q(Q)\} \\ &\geq w(Q) - (w(Q \cap S) + w(Q) - w_q(Q)) \\ &= w_q(Q) - w(Q \cap S) \end{aligned}$$

This can be further rewritten as

$$\begin{aligned} w(S) &\geq w_q(Q) - (w(Q \cap S) \\ w(Q \cap S) + w(S \setminus Q) &\geq w_q(Q) - w(Q \cap S) \\ 2w(Q \cap S) + w(S \setminus Q) &\geq w_q(Q) \end{aligned}$$

Hence, a vector of binary variables associated with the vertices $x_v,\ v \in V$ representing a weighted safe set must satisfy the following inequality

$$\sum_{v \in Q} 2w(v)x_v + \sum_{v \in V \setminus Q} w(v)x_v \geq w_q(Q). \tag{7}$$

2.2.2. Excluded Components

Let $G[C']$ be a connected subgraph and S a safe set of G. We denote by $\mathcal{X}$ the set of all connected components of $G[V(G) \setminus C']$. Next, we consider restrictions on S.

Lemma 3. *If $w(X) < w(C')$ for some $X \in \mathcal{X}$, then either $S \cap C' \neq \emptyset$ or $S \cap X = \emptyset$.*

Proof. Assume $S \cap C' = \emptyset$ and $\mathcal{X}$ contains one component X with $S \cap X \neq \emptyset$. Since X and C' are neighbors, $G[X \cap S]$ must have a connected component T which is neighbor of a connected component W of $G[V \setminus S]$ with $C' \subseteq W$. Because S is a safe set, $w(W) \leq w(T)$. Since all weights are non-negative, we have $w(C') \leq w(W) \leq w(T) \leq w(X)$. □

In order to extend the results of Lemma 3, let $\mathcal{L}$ be a subset of $\mathcal{X}$ of components which cannot intersect S if C' does not. By Lemma 3, we already know that any $X \in \mathcal{X}$ with $w(X) < w(C')$ can be included into $\mathcal{L}$.

Lemma 4. *If $Y \in \mathcal{X}$ (but $Y \notin \mathcal{L}$) and $w(Y) < w(C') + \sum_{X \in \mathcal{L}} w(X)$, then $Y \cap S = \emptyset$ if $S \cap C' = \emptyset$.*

Proof. Let W be the union of C' and every $X \in \mathcal{L}$. We can verify that $G[W]$ is a connected subgraph, contains no vertex in S, and $w(W) = w(C') + \sum_{X \in \mathcal{L}} w(X)$. Therefore, $w(Y) < w(W)$ and, by Lemma 3, $S \cap Y = \emptyset$ when $S \cap W = \emptyset$, which is the case when $S \cap C' = \emptyset$. □

Using the previous results, we introduce some notations. Let $\mathcal{K}^<$ be the subset of $\mathcal{X}$ including all components that must be disjoint from S when C' is disjoint from S. This set can be computed by first selecting all components satisfying the condition of Lemma 3 and then iteratively adding all components of $\mathcal{X}$ that satisfy the conditions of Lemma 4.

Theorem 5. *If the safe set S is such that $S \cap C' = \emptyset$, then $S \cap X = \emptyset$ for each $X \in \mathcal{K}^<$.*

Proof. It follows by construction and lemmas 3 and 4. □

Theorem 5 allows us to define valid inequalities for the model. Let $\mathcal{R}$ contain pairs $(C', \mathcal{K}^<)$.

$$|V| \sum_{v \in C'} x_v - \sum_{\substack{K \in \mathcal{K}^< \\ v \in K}} x_v \geq 0 \qquad\qquad (C', \mathcal{K}^<) \in \mathcal{R} \tag{8}$$

We also note that, for some C', it cannot be that $S \cap C' = \emptyset$.

Corollary 6. *If $\mathcal{X} = \mathcal{K}^<$, then $C' \cap S \neq \emptyset$.*

Let $\mathcal{I}$ be a set of components C' which must intercept every safe set. According to Corollary 6, we can add the following inequalities to the model.

$$\sum_{v \in C'} x_v \geq 1 \qquad\qquad C' \in \mathcal{I} \tag{9}$$

We will use some additional notation for the last inequality in this family. Let the total weight of $\mathcal{K}^<$ be $w^< = \sum_{X \in \mathcal{K}^<} w(X)$ and let $\mathcal{K}^> = \mathcal{X} \setminus \mathcal{K}^<$ be the set of components of $\mathcal{X}$ not in $\mathcal{K}^<$. Note that, every $K \in \mathcal{K}^>$ is such that $w(K) \geq w^<$.

Given a safe set S such that $C' \cap S = \emptyset$, consider the connected component W of $G[V \setminus S]$ that contains C'. For any $X \in \mathcal{X}$, we denote by $N(X, S, C')$ the set $W \cap X$ and let $w(X, S, C') = \sum_{v \in N(X,S,C')} w_v$.

Theorem 7. *If $Y \in \mathcal{K}^>$ and $w(Z) < w(C') + w(Y) + w^<$ for all $Z \in \mathcal{K}^>$, then either $S \cap C' \neq \emptyset$ or $w(Y \cap S) \geq w(C') + w^< + w(Y, S, C')$.*

Proof. Suppose $S \cap C' = \emptyset$ and there is a $Y \in \mathcal{K}^>$ satisfying the hypothesis of the theorem. Let W be the connected component of $G[V \setminus S]$ that contains C'. Every component in $\mathcal{K}^<$ should not intersect S by Lemma 3. So, W contains them all, along with the vertices in $N(Y, S, C')$. Hence, $w(W)$ is at least $w(C') + w^< + w(Y, S, C')$. Now, if $Y \cap S = \emptyset$, then $N(Y, S, C') = Y$ and $w(W)$ would be at least $w(C') + w(Y) + w^<$. Therefore, there must be a component of $G[S]$ with at least this much weight and it would have to be contained in one component of $\mathcal{K}^>$, but for each component $Z \in \mathcal{K}^>$, we have $w(Z) < w(C') + w(Y) + w^<$. So, $Y \cap S \neq \emptyset$ and, since Y is neighbor of C', $Y \cap S$ must contain a component that weighs at least $w(C') + w^< + w(Y, S, C')$. Note that this is also true, even if Y is the only component in $\mathcal{K}^>$. □

Theorem 7 provides a way to require the safe set to intersect and have a minimum weight on some connected components of G. As it is difficult to express $w(Y, S, C')$ as a function of S in a linear constraint, we took a relaxation of the weight requirement, considering that $w(Y, S, C') \geq w((N(C') \cap Y) \setminus S)$. This leads to the following inequality, using $\mathcal{N} = N(C') \cap Y$ and $\beta = w(C') + w^< + w(\mathcal{N})$.

$$\sum_{v \in Y} w_v x_v + \beta \sum_{v \in C'} x_v \geq \beta - \sum_{v \in \mathcal{N}} w_v x_v$$

Then, considering $\mathcal{F}$ the set with all pairs (C', Y) that satisfy the conditions on Theorem 7 and rearranging the terms, we get the following inequalities:

$$2 \sum_{v \in \mathcal{N}} w_v x_v + \sum_{v \in Y \setminus \mathcal{N}} w_v x_v + \beta \sum_{v \in C'} x_v \geq \beta \qquad (C', Y) \in \mathcal{F} \tag{10}$$

3. Branch-and-Cut Algorithm

Formulation (1)-(4) has an exponential number of Neighbor Components inequalities (3) and Connectivity constraints (5) (for the connected case), which can be explicitly enumerated only for very small graphs. Hence, we implemented a Branch-and-Cut Algorithm (B&C) to solve the formulation, where only a small subset of the inequalities (3) is initially considered, and additional violated inequalities as well as Connectivity constraints (5) are added on-the-fly during the computation, thanks to a separation procedure. The computational performance of B&C is further improved by adding clique inequalities (6) and Excluded Component inequalities (8), (9) and (10). Also, it is well known that providing an initial feasible solution of good quality and implementing primal heuristics at the nodes of the branch-and-bound tree can help in fathoming nodes and fasten the convergence of the algorithm. To this aim, algorithm B&C also embeds a primal heuristic algorithm. This section describes in detail all the procedures and algorithms needed to design algorithm B&C.

3.1. Separation

In this section we present separation procedures for the Neighbor Components inequalities (3) and Connectivity constraints (5), and for the strengthening inequalities introduced in the previous section.

Neighbor Components inequalities. We discuss separation of integer solutions. Let x^* be an integer solution and $S = \{v \in V,\ x^*_v = 1\} \subset V(G)$ the corresponding subset of vertices. The separation problem is to find a pair $(C, C') \in \mathcal{P}$ and a vertex $l \in C$ that violate inequality (3).

First we search for a pair $(C, C') \in \mathcal{P}$ that violates the condition of Lemma 2. We compute the collection of all connected components of $G[S]$, which we denote by $\mathcal{S}$; and the collection of all connected components of $G[V \setminus S]$, which we denote by $\mathcal{N}$. Connected components are identified in linear time by means of a DFS algorithm. Then, for each $C \in \mathcal{S}$ and each $C' \in \mathcal{N}$ with $w(C) < w(C')$, we check if there exists an edge $uv \in E$ such that $u \in C$ and $v \in C'$.

Every such pair (C, C') violating Inequality (3) would be sufficient be to separate the current solution. However, in order to strengthen the constraints that will be added to the formulation, each identified pair (C, C') is improved through a procedure which aims at reducing the number of vertices in $C' \cup N(C)$, i.e., the left hand side of the inequality. We consider two improving moves for each pair, which are iterated until no further improvement is viable. At every iteration, we try to improve the pair as follows:

- For each $v \in C'$, remove v from C', thus obtaining a new pair $(C, C' \setminus \{v\})$.
- For each $v \in N(C)$ such that $N(v) \subseteq C \cup C' \cup N(C)$, add v to C, thus obtaining a new pair $(C \cup \{v\}, C' \cup (N(v) \setminus C))$.

Among all possible moves defining a new pair $C, C' \in \mathcal{P}$ violating the conditions of Lemma 2, we select the move defining the pair maximizing $\sum_{v \in C'} w(v) - \sum_{v \in C} w(v)$.

After the improvement, an inequality (3) is generated for the selected pair, by choosing the vertex $l \in C \cap X$ with minimum weight.

Connectivity constraints. Given an integer solution x^* defining a disconnected safe set S, in order to separate connectivity constraints (5) we first identify two vertices $u, v \in S$ that belong two different connected components of G, say C_u and C_v. After removing from G all edges in $E[C_u \cup N(C_u)]$, we define R_v as the set of vertices that can be reached from v in the resulting graph. A minimal separator is then $N(C_u) \cap R_v$. In practice, R_v can be computed in linear time by a BFS algorithm on G starting from vertex v, and never expanding the search from a vertex in $N(C_v)$. We refer the reader to [7] for more details.

A separation procedure for fractional solutions through the computation of a max-flow among each pair of vertices of G is also discussed in [7]. However, the authors observe that this is a time consuming procedure and does not improve the computational performance when looking for Steiner trees. We observed a similar behaviour in our preliminary computational experiments on the connected Safe Set Problem, and hence decided to only separate integer solutions.

Clique Cuts. For separating Clique inequalities (6), we propose a separation procedure that works both for integer and fractional solutions. Given a (possibly fractional) solution x^*, the separation problem is to find a clique K^* for which (6) are violated, that is

$$2 \sum_{v \in K^*} w_v x^*_v + \sum_{v \in V \setminus K^*} w_v x^*_v < w(K^*).$$

This is equivalent to

$$\sum_{v \in K^*} w_v (1 - 2x^*_v) - \sum_{v \in V \setminus K^*} w_v x^*_v > 0.$$

Since $\sum_{v \in V \setminus K^*} w_v x^*_v = \sum_{v \in V} w_v x^*_v - \sum_{v \in K^*} w_v x^*_v$, the inequality can be rewritten as

$$\sum_{v \in K^*} w_v (1 - x^*_v) > \sum_{v \in V} w_v x^*_v. \tag{11}$$

Given x^*, the right hand side of (11) is a constant. The separation problem is then to find a maximum weight clique of G, with weights defined as $w_v(1-x^*_v)$ for each $v \in V$. Since finding a maximum weight clique is NP-hard, this problem is solved using the *Multi-neighborhood Tabu Search* heuristic algorithm described in [11].

Unfortunately for inequalities (7), the right hand side is not a simple sum of weights for all the vertices in Q, hence the above derivation is not valid.

Algorithm 1 Separation heuristic for Excluded Components Cuts

function SEPARATE-EXCLUDED-COMPONENTS(G, S)
  Initialize $F \leftarrow \emptyset$ ▷ Pool of found inequalities.
  for each (C, C') violating a Neighbor Component inequality **do**
    Store vertices of C' by non-decreasing weight into a list O
    repeat
      Compute $\mathcal{X}$, $\mathcal{K}$, $\mathcal{K}^<$, $\mathcal{K}^>$, and $w^<$ for C'
      if $\mathcal{K}^> = \emptyset$ **then**
        Add to F inequality (9) from Corollary 6
      if there is a $K \in \mathcal{K}^<$ such that $K \cap S \neq \emptyset$ **then**
        Add to F inequality (8) from Theorem 5
      for each $Y \in \mathcal{K}^>$ **do**
        if $w(Y) + w(C') + w^< > w(Z)$ for every $Z \in \mathcal{K}^>$, $w(Y \cap S) < w(C') + w^< + w(Y, S, C')$ **then**
          Add to F inequality (10) from Theorem 7
      updated $\leftarrow$ false
      repeat
        Remove the first vertex v from O
        if v is not a cut vertex of $G[C']$ **then**
          Set $C' \leftarrow C' \setminus \{v\}$
          updated $\leftarrow$ true
      until updated or O is empty
    until not updated
  return F

Excluded Components. Given an integer solution, we use Algorithm 1 to look for violated Excluded Component inequalities. The algorithm considers each components C' identified when separating Neighbor Component inequalities, and checks if the same component induces Excluded Component inequalities as well. After that, it tries to remove vertices from C' without disconnecting the component, and iterates the check. All found inequalities are stored in a pool $\mathcal{F}$. If the algorithm finds several violated inequalities, we add to the model at most 5 of each type for each initial component C'. We give priority to inequalities with largest violation.

It is also possible to separate fractional solutions by using the same algorithm. One needs only to define $S = \{v : x^*_v > 0\}$ for the solution x^* and redefine $w(Y \cap S) = \sum_{v \in Y \cap S} w_v x^*_v$ and $w(Y, S, C') = \sum_{v \in N(Y,S,C')} w_v x^*_v$. In practice, however, separating fractional solutions did not produce any noticeable improvement.

3.2. Primal heuristic

As primal heuristic we adapt the initial heuristic proposed by [5]. We design an algorithm which can be used both as node heuristic and initial heuristic, and it is described in Algorithm 2. The heuristic defines a minimal weighted

safe set for G containing all vertices from the *Required* set and no vertex from the *Forbidden* set. The algorithm defines an initial set Sol including all non-forbidden vertices. If this set is not a safe set of G, the heuristic fails. Otherwise, the algorithm iterates by picking a vertex in a restricted candidate list of size k, where vertices are initially sorted according to a label L_v that is initialized with the vertex degree in G. The vertex is removed from Sol if this operation does not spoil the safe set; in this case the label L_v of each vertex in the Candidate list is increased by the weight of the removed vertex.

When the algorithm is invoked as a initial heuristic, the required and forbidden sets are empty; when it is invoked as node heuristic, the required and forbidden sets contain all vertices whose variables are fixed at one and zero at the node, respectively.

Algorithm 2 Weighted safe set heuristic with restrictions

function WSSP-HEURICTIC($G, k, Required, Forbidden$)
  Set $Sol \leftarrow V(G) \setminus Forbidden$
  if Sol is not a weighted safe set of G **then**
    return `false`, Sol
  Set $Candidates \leftarrow Sol \setminus Required$
  Set $L_v \leftarrow d_G(v)$ for each $v \in Candidates$
  while $Candidates \neq \emptyset$ **do**
    Select a random vertex v from the k vertices of $Candidates$ with least L_v
    Remove v from $Candidates$
    if $Sol \setminus \{v\}$ is a weighted safe set of G **then**
      Remove v from Sol
      for all $x \in N(v)$ **do**
        Set $L(x) \leftarrow L(x) + w(v)$
  return `true`, Sol

4. Computational experiments

This section provides details on the implementation and on the computational performance of our B&C algorithm. By considering two large set of instances from the literature, we are also able to compare B&C with two recently proposed approaches.

4.1. Implementation details

The B&C algorithm starts by invoking 50 times the heuristic of Section 3.2, so as to have an initial feasible solution of good quality. Later, the heuristic is invoked every 60 nodes of the branch-and-bound search tree. The full version of B&C performs separation at all integer solutions. As soon as an inequality is found which is violated by the current solution, the linear program is re-optimized. Inequalities are considered in this order:

1. Excluded Components inequalities;
2. Clique inequalities - separated every 8 nodes only, up to node 8,000;
3. Neighbor Components inequalities.

For the connected Safe Set Problem, connectivity constraints are separated before the incumbent solution is updated.

In addition, Clique inequalities are separated also for fractional solutions every 8 nodes up to node 8,000.

The B&C algorithm was implemented within the SCIP solver [12] version 6.0.2 along with the Soplex LP solver version 4.0.2. All our experiments were executed on a machine with 32GB RAM and a i7-4790 processor running at 3.60GHz. The solver was used in single thread mode with maximum memory usage limited to 28 GB. The program source was written in C/C++ and compiled with g++ version 8.3.0.

We also tested a variant of B&C implemented within the Gurobi branch-and-cut framework. Although Gurobi is on average able to solve the same considered instances in a smaller number of branch-and-bound nodes, the time per node is increased of one order of magnitude. Overall, the performance of the resulting algorithm is inferior. Hence, in what follows, we report results for the SCIP implementation of B&C.

Instances. As for the test instances, we considered two sets of instances from the recent literature.

A first set includes 63 pseudo-random graphs introduced by Macambira et. al. [5]. For these instances the cardinality n of the vertex set, ranging between 10 and 30, and the density D, equal to 0.3, 0.5 and 0.7, are specified. Each instance contains a random spanning tree, which is then extended by adding random edges until the desired density is reached. For the weighted Safe Set Problem, a weight randomly selected between 1 and 100 is associated with each vertex of the graph. Results for these instances are also reported by Hosteins [6].

A second set includes 120 instances introduced by Hosteins [6]. For these instances the cardinality n of the vertex set is in $\{20, 25, 30, 35, 40, 50\}$ with different possible values for the edge density D in $\{0.1, 0.2, 0.3, 0.4\}$. For each combination, the set includes 5 randomly generated instances contains a random spanning tree, which is then extended by adding random edges, similar to the first set.

4.2. Effect of strengthening inequalities

In a first set of experiment we are interested in evaluating the improvement on the performance of the B&C algorithm that is due to the addition of Clique inequalities and Neighbor Components inequalities. By disabling all the strengthening inequalities we obtain a basic version of B&C only implementing Neighbor Component inequalities. We denote this variant as NCC. An intermediate version, obtained by including Clique inequalities, is denotes as NCC+CC. The full version of B&C is obtained by activating Excluded Component inequalities and is denoted as NCC+CC+ECC. We tested these three

		NCC			NCC+CC			NCC+CC+EC		
n	D	Time	(K)Nodes	(K)Cuts	Time	(K)Nodes	(K)Cuts	Time	(K)Nodes	(K)Cuts
10	0.3	0.0	0.06	0.2	0.0	0.2	0.4	0.0	0.06	0.1
15	0.3	0.3	1.3	5.0	0.2	1.7	5.4	0.1	0.5	0.7
20	0.3	23.2	81.8	237.9	6.6	25.1	191.2	0.4	4.1	5.0
25	0.3	444.3	1,284.2	4,149.7	146.9	424.8	3,161.1	1.6	9.2	13.3
30	0.3	**7,200.0**	13,706.7	23,386.1	4,918.8	13,711.1	33,902.1	14.7	91.9	110.2
15	0.5	0.7	3.2	9.1	0.6	4.0	10.0	0.1	1.0	1.2
20	0.5	35.1	102.0	312.0	14.7	67.0	268.7	0.2	2.0	2.5
25	0.5	1,106.2	2,666.0	9,545.1	584.4	1,959.4	9,086.8	2.7	24.0	28.1
30	0.5	**6,046.8**	8,287.0	24,514.7	**6,308.9**	19,597.1	36,709.4	65.9	690.9	760.3
10	0.7	0.0	0.2	0.4	0.0	0.4	0.7	0.0	0.2	0.2
15	0.7	1.1	4.6	13.5	1.0	7.4	16.1	0.1	1.4	1.7
20	0.7	48.9	131.8	406.0	20.5	98.6	286.8	0.5	5.6	6.2
25	0.7	1,569.8	3,399.2	12,568.1	829.8	2,691.0	10,388.7	2.6	27.4	30.6
30	0.7	**5,287.1**	7,016.7	22,064.6	**6,717.4**	18,114.0	50,397.4	46.8	442.5	489.3

Table 1: Time and branch-and-bound nodes and cuts of three variants of B&C when solving the Weighted Safe Set Problem on a subset of instances.

variants of B&C on a subset of the instances from [5] having a number n of vertices in $\{10, 15, 20, 25, 30\}$. We considered the Weighted Safe Set Problemand limited the run time to 7,200 seconds.

In Table 1, we report, for the considered subset of 15 instances, number of vertices n, density D, and the computing time, number of branch-and-bound nodes (thousands) and added inequalities (thousands) for each variant of B&C. Some runs stopped before 7,200 seconds without solving the problem due to the memory limit; we emphasized the run time of non-solved instances in black in the table. The basic algorithm NCC can solve all but three instances within two hours. For all graph densities, the computing time, the number of branch-and-bound nodes and added inequalities grow very fast when graphs with 25 and more vertices are considered. When adding the Clique inequalities, variant NCC+CC can solve all but two instances and is able to reduce the computing time for the large graphs in the set, however, the number of branch-and-bound nodes can be as large as almost 20 millions and the number of inequalities is still huge. The effect of separating Excluded Component inequalities in variant NCC+CC+ECC is dramatic. The computing time for the largest instances with 30 vertices is reduced by two orders of magnitude. The number of branch-and-bound nodes is reduced between one and two orders of magnitude, and at most $760K$ inequalities are added for the hardest instance ($n = 30$, $D = 0.5$).

4.3. Comparison with state-of-the-art

The recent literature on the Safe Set Problem and Weighted Safe Set Problem include two different approaches. The first algorithm is a Branch-and-Cut algorithm [5] based on a formulation with an exponential number of constraints. The second approach is based on the solution of the compact model described in [6]. The experiments reported in this paper show that this second formulation

outperforms the approach in [5]. Therefore, we consider the state-of-the-art approach in [6] as a reference for evaluating the performance of the B&C algorithm (in its full variant).

n	D	Unweighted			Unweighted [6]			Weighted			Weighted [6]		
		Time	Gap	Sol.	Time	Gap	Sol.	Time	Gap	Sol.	Time	Gap	Sol.
20	0.1	1.3	0.0	5	94.2	0.0	5	0.8	0.0	5	72.2	0.0	5
25	0.1	5.2	0.0	5	532.2	0.0	5	4.9	0.0	5	746.4	0.0	5
30	0.1	32.4	0.0	5	1,743.8	4.4	4	20.1	0.0	5	2,554.6	16.7	3
35	0.1	469.4	0.0	5	3,538.0	33.8	1	55.0	0.0	5	3,600.0	39.5	0
40	0.1	2,976.1	19.0	1	3,600.0	67.7	0	1,244.2	2.2	4	3,600.0	74.9	0
50	0.1	3,600.0	37.6	0	3,600.0	88.0	0	3,600.0	22.7	0	3,600.0	94.6	0
				21			15			24			13
20	0.2	0.6	0.0	5	15.4	0.0	5	0.3	0.0	5	28.0	0.0	5
25	0.2	4.9	0.0	5	234.4	0.0	5	1.9	0.0	5	140.2	0.0	5
30	0.2	45.2	0.0	5	856.8	0.0	5	10.6	0.0	5	1,472.0	7.5	4
35	0.2	2,825.1	7.7	2	3,437.0	18.0	2	195.3	0.0	5	3,522.6	34.5	1
40	0.2	3,116.9	29.2	0	3,600.0	71.1	0	1,207.8	0.0	5	3,600.0	72.2	0
50	0.2	3,600.0	45.8	0	3,600.0	93.1	0	3,600.0	33.1	0	3,600.0	93.7	0
				17			17			25			15
20	0.3	0.2	0.0	5	14.4	0.0	5	0.2	0.0	5	11.8	0.0	5
25	0.3	12.5	0.0	5	77.6	0.0	5	2.3	0.0	5	52.0	0.0	5
30	0.3	174.3	0.0	5	362.6	0.0	5	22.5	0.0	5	562.2	0.0	5
35	0.3	3,141.5	10.5	1	2,798.2	10.6	3	326.6	0.0	5	1,956.8	4.3	4
40	0.3	3,368.4	30.3	0	3,600.0	38.4	0	2,400.3	2.1	3	3,384.0	43.2	1
50	0.3	3,600.0	43.2	0	3,600.0	91.1	0	3,600.0	24.1	0	3,600.0	96.4	0
				16			18			23			20
20	0.4	1.1	0.0	5	14.2	0.0	5	0.5	0.0	5	12.8	0.0	5
25	0.4	40.8	0.0	5	98.8	0.0	5	5.1	0.0	5	66.4	0.0	5
30	0.4	256.6	0.0	5	267.8	0.0	5	34.2	0.0	5	375.0	0.0	5
35	0.4	3,486.8	9.8	1	1,709.4	2.4	4	404.6	0.0	5	905.4	0.0	5
40	0.4	3,600.0	26.6	0	2,678.4	11.6	4	1,367.2	0.0	5	2,788.0	16.1	3
50	0.4	3,600.0	39.6	0	3,600.0	95.3	0	3,600.0	17.8	0	3,600.0	99.4	0
				16			23			25			23

Table 2: Results for SSP and WSSP on instances from [6]

.

In Table 2, we consider the results obtained by B&C and by the approach in [5] when solving the (unconnected) Safe Set Problem and Weighted Safe Set Problem. Results for the approach in [5] are obtained on a computer similar to the one we are using in our experiments and are borrowed from that paper. We also used the same time limit of 3,600 seconds. Each row of the table is relative to 5 instances, for which we report the number of vertices and density. Then, for each algorithm, we report the computing time, percentage optimality gap (computed as absolute gap over best solution value) and number of optimally solved instances (out of five). The left part of the table is relative to the Safe Set Problem, the right part to the Weighted Safe Set Problem. Horizontal lines

separate groups of instances of the same density. We also report the sum of optimally solved instances for each group.

Concerning the (unweighted) Safe Set Problem, the two approaches are equivalent for densities of 0.2 and 0.3, while B&C has a better performance for low density instances (0.1), and the approach in [6] has a better performance for high density instances (0.4). For small graphs ($n = 20, 25$) both approaches can solve all instances, but B&C is often faster of about one order of magnitude. Overall, algorithm B&C solves 70 unweighted instances, while the approach in [6] has a sligtly better performance as it can solve 73 instances to optimality.

When tackling the Weighted Safe Set Problem, B&C improves its relative performance and it is the best approach for all considered classes of density. It can solve almost all instances with up-to 40 vertices, and the average optimally gap for instances with 50 vertices is at most 33.1%. The approach in [6] cannot solve most of the instances with $n = 40$, and none with $n = 50$. For this last group, optimality gaps are always larger than 93.7%. Overall, algorithm B&C solves 93 weighted instances, outperforming the approach in [6] which can solve solve 71 instances to optimality.

In Table 3, we consider the results obtained by B&C and by the approach in [5] when solving the connected Safe Set Problem and the connected Weighted Safe Set Problem. The table has the same structure of Table 2.

The performance of the two approaches is similar to the one observed in the unconnected case. Concerning the Safe Set Problem, the two approaches are approximately equivalent for instances of average density; B&C has a better performance for low density instances (0.1), and the approach in [6] has a better performance for high density instances (0.4). Overall, algorithm B&C solves 72 unweighted instances, while the approach in [6] has a better performance as it can solve 76 instances to optimality.

Concerning the Weighted Safe Set Problem, B&C is the best algorithm for densities up-to 0.4. It can solve 24 instances having density 0.4, while the approach in [6] can solve one more instance in this class. No approach can solve to optimality instances with $n = 50$. Overall, algorithm B&C solves 96 weighted instances, outperforming the approach in [6] which can solve 84 instances to optimality.

Finally, Table 4 considers the instances proposed in [5] for the unconnected versions of the Safe Set Problem and Weighted Safe Set Problem. Each row of the table is relative to one value of n (number of vertices). Then, for each considered density, the table reports the computing time of B&C and [5], respectively. The left part of the table is for the unweighted case, the right part for the weighted case.

Both the B&C algorithm and the approach considered in [6] can solve all these 120 instances. These instances were also solved with the Branch-and-Cut algorithm in [5]. This algorithm can solve only 58 instances when considering the Safe Set Problem, and 57 instances when considering the Weighted Safe Set Problem, respectively. For this algorithm, the computing times for solved instances are approximately one order of magnitude larger than those of [6].

n	D	Unweighted			Unweighted [6]			Weighted			Weighted [6]		
		Time	Gap	Sol.	Time	Gap	Sol.	Time	Gap	Sol.	Time	Gap	Sol.
20	0.1	0.6	0.0	5	4.4	0.0	5	0.8	0.0	5	4.6	0.0	5
25	0.1	4.0	0.0	5	41.4	0.0	5	5.3	0.0	5	55.0	0.0	5
30	0.1	14.9	0.0	5	258.4	0.0	5	16.9	0.0	5	284.4	0.0	5
35	0.1	239.8	0.0	5	2,423.0	29.3	2	28.5	0.0	5	1,614.2	11.1	4
40	0.1	2,912.0	13.7	2	3,492.8	38.6	1	941.3	0.0	5	3,529.6	42.0	1
50	0.1	3,600.0	42.1	0	3,600.0	66.3	0	3,600.0	18.0	0	3,600.0	74.8	0
				22			18			25			20
20	0.2	0.4	0.0	5	7.8	0.0	5	0.2	0.0	5	6.8	0.0	5
25	0.2	2.6	0.0	5	47.2	0.0	5	1.8	0.0	5	35.4	0.0	5
30	0.2	48.8	0.0	5	232.2	0.0	5	10.2	0.0	5	193.8	0.0	5
35	0.2	2,901.6	6.1	3	3,206.2	17.2	2	179.0	0.0	5	1,687.4	5.6	4
40	0.2	3,491.7	28.7	0	3,600.0	42.4	0	1,408.9	1.6	4	3,600.0	41.4	0
50	0.2	3,600.0	47.2	0	3,600.0	67.7	0	3,600.0	34.6	0	3,600.0	76.9	0
				18			17			24			19
20	0.3	0.4	0.0	5	7.2	0.0	5	0.2	0.0	5	6.4	0.0	5
25	0.3	9.4	0.0	5	48.6	0.0	5	2.4	0.0	5	51.2	0.0	5
30	0.3	169.9	0.0	5	256.4	0.0	5	24.6	0.0	5	217.6	0.0	5
35	0.3	2,825.2	8.8	2	2,265.2	0.0	5	321.4	0.0	5	1,300.6	0.0	5
40	0.3	3,386.5	28.9	0	3,600.0	45.3	0	2,352.1	2.9	3	3,600.0	45.4	0
50	0.3	3,600.0	43.8	0	3,600.0	92.7	0	3,600.0	24.9	0	3,600.0	83.5	0
				17			20			23			20
20	0.4	1.0	0.0	5	9.6	0.0	5	0.4	0.0	5	9.8	0.0	5
25	0.4	38.9	0.0	5	71.8	0.0	5	4.3	0.0	5	74.8	0.0	5
30	0.4	210.4	0.0	5	236.0	0.0	5	49.4	0.0	5	302.4	0.0	5
35	0.4	3,600.2	9.9	0	2,485.0	2.6	4	441.5	0.0	5	1,002.4	0.0	5
40	0.4	3,600.1	24.3	0	3,297.0	18.9	2	1,922.9	1.1	4	2,915.0	0.0	5
50	0.4	3,600.0	39.7	0	3,600.0	92.6	0	3,600.0	15.8	0	3,600.0	95.2	0
				15			21			24			25

Table 3: Results for connected SSP and connected WSSP on instances from [6]

.

Comparing B&C and the approach in [6], the table shows the better performance of B&C for densities 0.3 and 0.5. Indeed, our algorithm is at least two orders of magnitude faster in the unweighted case and approximately three orders of magnitude faster in the weighted case, respectively. For density 0.7, the method in [6] is faster in the unweighted case, while B&C is faster in the weighted case. These results are in line with the experiments reported in Tables 2 and 3.

5. Conclusion

We have proposed a new integer linear formulation for the Weighted Safe Set Problem. The formulation uses a minimal set of variables, associated with the vertices that are included in the safe set, while the structure of the safe set

	Unweighted						Weighted					
D	0.3		0.5		0.7		0.3		0.5		0.7	
n	t	[6]	t	[6]	t	[6]	t	[6]	t	[6]	t	[6]
11	0.0	0	0.2	1	19.6	0	0.0	1	0.1	0	1.4	1
12	0.0	1	0.1	1	4.3	1	0.0	1	0.1	0	7.3	1
13	0.0	1	0.0	1	27.5	1	0.0	1	0.0	1	1.6	1
14	0.0	2	0.2	2	38.8	2	0.0	2	0.1	2	2.7	3
15	0.0	2	0.3	3	10.1	4	0.0	2	0.1	3	2.6	4
16	0.0	2	0.8	4	33.1	6	0.0	4	0.2	4	7.2	6
17	0.0	5	1.5	6	455.9	4	0.0	4	0.1	4	4.3	4
18	0.0	7	2.6	9	80.7	6	0.0	4	0.5	7	2.8	10
19	0.0	10	0.5	10	24.9	12	0.0	13	0.4	10	5.8	14
20	0.0	11	0.8	15	93.3	22	0.0	11	0.2	13	25.4	19
21	0.0	16	0.7	27	140.0	37	0.0	11	0.5	20	9.6	33
22	0.0	18	1.0	15	52.1	27	0.0	22	0.1	17	1.9	51
23	0.0	34	3.7	41	418.1	94	0.1	19	0.7	26	12.0	44
24	0.0	38	0.2	58	311.0	110	0.0	69	1.6	59	7.2	76
25	0.1	59	0.5	44	80.7	123	0.1	61	1.2	49	13.3	50
26	0.1	78	6.5	97	878.8	148	0.1	203	1.4	80	58.4	113
27	0.0	150	3.6	134	2,416.3	168	0.1	71	1.6	83	26.6	262
28	0.1	284	3.6	248	111.2	194	0.0	221	0.4	282	14.7	251
29	0.1	182	11.1	130	2,418.2	178	0.0	138	0.8	211	65.9	254
30	0.1	472	11.1	359	439.1	332	0.2	219	0.1	267	46.8	305

Table 4: Times for SSP and WSSP for Macambira instances.

is imposed via an exponential number of neighbor component inequalities and connectivity constraints. The proposed formulation has fewer variables than any other model in the literature, and can be further strengthened by additional inequalities, that we denotes as clique and excluded component inequalities. We presented separation procedures for the introduced inequalities, which allowed us to design a full branch-and-cut algorithm for solving the formulation. The algorithm was tested on a sets of benchmark instances from the literature, in four variants of the problem, which arise by imposing connectivity of the safe set, and by considering weighted or unweighted vertices, respectively. Our computational experiments showed that our branch-and-cut algorithm is particularly effective on low and medium density instances and in the weighted case, where it outperforms the state-of-the-art approach from the literature, still having a comparable performance on dense unweighted instances.

References

[1] S. Fujita, G. MacGillivray, T. Sakuma, Safe set problem on graphs, Discrete Applied Mathematics 215 (2016) 106–111.

[2] R. Bapat, S. Fujita, S. Legay, Y. Manoussakis, Y. Matsui, T. Sakuma,

Z. Tuza, Network majority on tree topological network, Electron. Notes Discrete Math. 54 (2016) 79–84.

[3] R. Águeda, N. Cohen, S. Fujita, S. Legay, Y. Manoussakis, Y. Matsui, L. Montero, R. Naserasr, H. Ono, Y. Otachi, T. Sakuma, Z. Tuza, R. Xu, Safe sets in graphs: graph classes and structural parameters, J. Comb. Optim. 36 (4) (2018) 1221–1242.

[4] R. Belmonte, T. Hanaka, I. Katsikarelis, M. Lampis, H. Ono, Y. Otachi, Parameterized complexity of safe set, Journal of Graph Algorithms and Applications 24 (2020) 215–245.

[5] A. F. U. Macambira, L. Simonetti, H. Barbalho, P. H. Gonzalez, N. Maculan, A new formulation for the safe set problem on graphs, Computers and Operations Research 111 (2019) 346–356.

[6] P. Hosteins, A compact mixed integer linear formulation for safe set problems, Optimization Letters 14 (2020) 2127–2148.

[7] M. Fischetti, M. Leitner, I. Ljubić, M. Luipersbeck, M. Monaci, M. Resch, D. Salvagnin, M. Sinnl, Thinning out steiner trees: a node-based model for uniform edge costs, Mathematical Programming Computation 9 (2017) 203–229.

[8] F. Furini, I. Ljubić, E. Malaguti, P. Paronuzzi, On integer and bilevel formulations for the k-vertex cut problem, Mathematical Programming Computation 12 (2019) 133–164.

[9] F. Furini, I. Ljubić, E. Malaguti, P. Paronuzzi, Casting light on the hidden bilevel combinatorial structure of the k-vertex separator problem, Operations Research (2021). doi:10.1287/opre.2021.2110.

[10] W. Ben-Ameur, M. D. Biha, On the minimum cut separator problem, Networks 59 (2012) 30–36.

[11] W. Qinghua, J.-K. Hao, F. Glover, Multi-neighborhood tabu search for the maximum weight clique problem, Annals of Operations Research 196 (2012) 611–634.

[12] G. Gamrath, T. Fischer, T. Gally, A. M. Gleixner, G. Hendel, T. Koch, S. J. Maher, M. Miltenberger, B. Müller, M. E. Pfetsch, C. Puchert, D. Rehfeldt, S. Schenker, R. Schwarz, F. Serrano, Y. Shinano, S. Vigerske, D. Weninger, M. Winkler, J. T. Witt, J. Witzig, The scip optimization suite 3.2, Tech. Rep. 15-60, ZIB, Takustr.7, 14195 Berlin (2016).