A Branch-and-Cut-and-Price Algorithm for Vertex-Biconnectivity Augmentation

Ivana Ljubić*

Faculty of Business, Economics and Statistics
University of Vienna
Brünnerstr. 72, 1210 Vienna, Austria
ivana.ljubic@univie.ac.at

Abstract. In this paper, the first approach for solving the vertex-biconnectivity augmentation problem (V2AUG) to optimality is proposed. Given a spanning subgraph of an edge-weighted graph, we search for the cheapest subset of edges to augment this subgraph in order to make it vertex-biconnected. The problem is reduced to the augmentation of the corresponding block-cut tree [18] whose connectivity properties are exploited to develop two minimum-cut-based ILP formulations: a directed and an undirected one. In contrast to the recently obtained result for the more general vertex-biconnected Steiner network problem [2], our theoretical comparison shows that orienting the undirected graph does not help in improving the quality of lower bounds. Hence, starting from the undirected cut formulation, we develop a branch-and-cut-and-price algorithm (BCP) which represents the first exact approach to V2AUG. Our computational experiments show the practical feasibility of BCP: Complete graphs with more than 400 vertices can be solved to provable optimality. Furthermore, BCP is even faster than the state-of-the-art metaheuristics and approximation algorithms, as far as graphs with up to 200 vertices are considered. For large graphs with more than 2000 vertices, optimality gaps that are strictly below 2% are reported.

Keywords: branch-and-cut; branch-and-cut-and-price; edge-weighted biconnectivity augmentation; memetic algorithm; generalized Steiner network problem

* Supported by the Hertha-Firnberg Fellowship of the Austrian Science Foundation (FWF)

1 The Vertex-Biconnectivity Augmentation Problem

In the design of modern telecommunication networks, in particular backbones, survivability is an important issue. In many telecommunication applications it is not acceptable that the failure of a single service node—a multiplexer, switch, or router, for example—leads to a disconnection of other nodes. Redundant connections need to be established to provide alternative routes in case of a temporary malfunction of any one node.

This kind of network redundancy is described in graph theory by means of *vertex connectivity*. A graph is said to be k-vertex connected, $k \geq 2$, if for every pair of distinct, non-adjacent vertices u and v, there are at least k vertex-disjoint paths (except for u and v) connecting them. In this paper we will often refer to 2-vertex connectivity as vertex-biconnectivity, or simply as biconnectivity, whenever it is clear from context.

In the *vertex-biconnectivity augmentation problem for edge-weighted graphs* (V2AUG), a spanning but not vertex-biconnected graph is given. Thus, the removal of a vertex may *disconnect* the graph into several connected components. We say that we *cover a vertex* when we add some edges to ensure that the removal of this vertex no longer disconnects the graph. The global aim is to identify a set of additional edges at minimum total cost in order to cover all vertices.

In this paper we will refer to the problem briefly as (V2AUG). Formally, the problem is defined as follows.

Definition 1. [V2AUG]
Let $G = (V, E)$ be a vertex-biconnected, undirected graph. Each edge $e \in E$ has an associated cost $c_e > 0$. A connected, spanning, but not vertex-biconnected subgraph $G_0 = (V, E_0)$, $E_0 \subset E$, is given. The edges $E_a = E \setminus E_0$ may be used for augmentation. The objective is to determine a subset $E'_a \subseteq E_a$ such that the augmented graph $G' = (V, E_0 \cup E'_a)$ is vertex-biconnected and the function

$$c(E'_a) = \sum_{e \in E'_a} c_e \tag{1}$$

is minimized.

Eswaran and Tarjan [5] were the first to investigate V2AUG. Using a reduction from the Hamiltonian circuit problem, they proved that the decision problem associated with V2AUG is NP-complete. An exact polynomial-time algorithm that runs in $O(|V| + |E|)$ time could be found for the special case when G has unit edge costs [11]. The best approximation ratio of 2 has been achieved by Khuller and Thurimella [18]; see also Khuller et al. [17]. A genetic algorithm was

given in [19] and was later improved by Ljubić and Raidl [20] by incorporating advanced local search and preprocessing techniques. In this paper, we will refer to this latter memetic algorithm as MA.

Considerable work has been done on the related edge-biconnectivity augmentation problem (E2AUG), a thorough review of which can be found in a recent work of Bang-Jensen et al. [1]. As this is the case in many problems related to graph biconnectivity, an application of algorithms developed for edge-biconnectivity does not lead to appropriate solutions for vertex-biconnectivity. In particular, Bang-Jensen et al. [1] model E2AUG using a compact ILP model based on a set covering formulation. The corresponding set covering ILP model for V2AUG would not be compact anymore, but would involve an exponential number of variables. To solve such a model, one might need to develop a sophisticated column generation algorithm.

Our Contribution. As our exact approach relies on the augmentation of the block-cut tree, which does not apply to E2AUG, the paper is focused on vertex-biconnectivity. In Section 2, the connectivity properties of that tree are studied and used to develop two minimum-cut-based integer linear programming (ILP) formulations of the problem: a directed and an undirected one. The former works on the undirected block-cut graph, the latter uses a new orientation-based characterization of the block-cut graph. This characterization is an extension of the general orientability property for biconnected graphs recently proposed by Chimani et al. [3]. The main (and surprising) result of this theoretical part of the paper is the proof that the orientation-based model of the augmented block-cut tree does not lead to stronger lower bounds than the undirected one.

Therefore, starting from the undirected-cut model, we develop a branch-and-cut (BC) algorithm whose main properties are given in Section 3.1. The main contribution of the practical part of the paper is the integration of sparse and reserve graph pricing techniques into BC that are described in detail in Section 3.2. A detailed computational analysis of the influence of pricing combined with BC is given in Section 4. We show that, apart from pricing, another two important features significantly improve the performance of the BC algorithm: a) the primal heuristic, and b) the initialization with high-quality upper bounds obtained after running a metaheuristic described in [20].

This is the first exact algorithm for solving V2AUG. Our branch-and-cut-and-price (BCP) algorithm is even faster than the state-of-the-art metaheuristics and approximation algorithms, as far as graphs with up to 200 vertices are considered. Complete graphs with more than 400

vertices are solved to provable optimality. For very large graphs (with more than 2000 vertices) we obtain lower bounds that are no more than 2% below the best known feasible solutions.

The Block-Cut Graph. All maximal subgraphs of a graph G_0 that are vertex-biconnected, i.e., its *vertex-biconnected components*, are referred to as *blocks*. As we assume that the graph G_0 is not vertex-biconnected, there will be at least two blocks in G_0. Any two blocks of G_0 share at most a single vertex, which we call a *cut-vertex*—the removal of a cut-vertex disconnects G_0 into several connected components. A *block-cut tree* $T = (V_T, E_T)$, with vertex set V_T and edge set E_T, is an undirected tree that reflects the relations between blocks and cut-vertices of graph G_0 in a simpler way (see, e.g., [5, 10]). The vertices of the block-cut tree $T = (V_T, E_T)$ are partitioned into the sets of cut- and block-vertices, $V_C \subset V_T$ and $V_B \subset V_T$, respectively. Figure 1(b) illustrates an example of a block-cut tree.

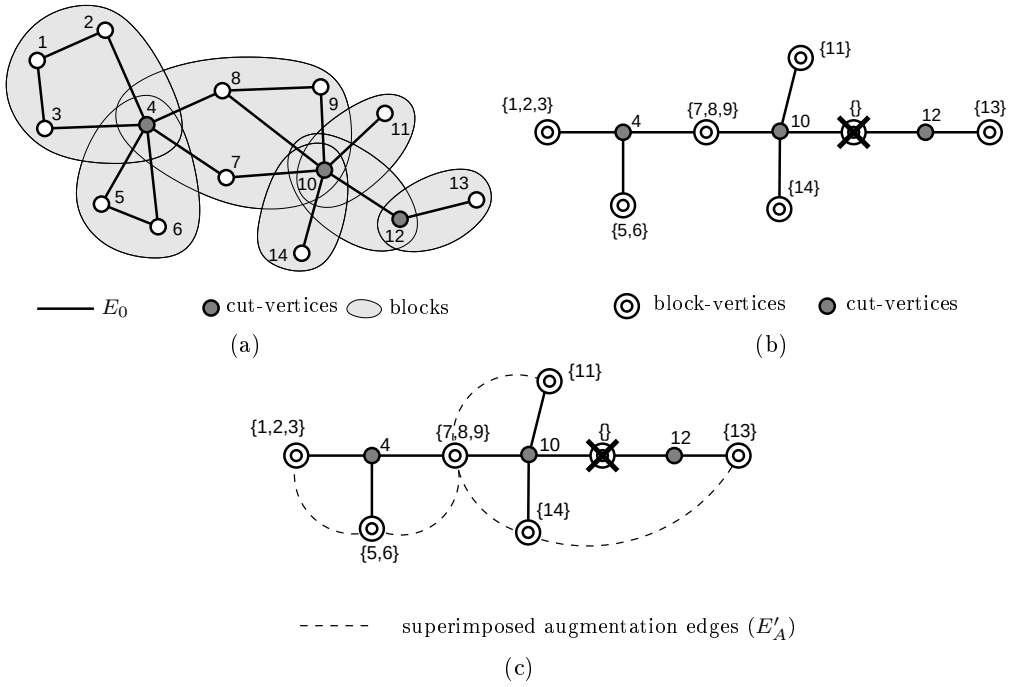


Fig. 1. (a) A connected, but not vertex-biconnected graph $G_0 = (V, E_0)$; (b) The corresponding block-cut tree $T = (V_T, E_T)$; (c) A feasible augmentation: the block-cut graph $G'_{BC} = (V_T, E_T \cup E'_A)$.

A block-vertex is associated to all vertices of the represented block in G_0 excluding cut-vertices. A cut-vertex $v_c \in V_C$ and a block-vertex $v_b \in V_B$ are connected by an undirected edge $\{v_c, v_b\}$ in E_T if and only if the cut-vertex corresponding to v_c in G_0 is part of the block represented by

v_b. Obviously, the resulting structure is always a tree. The computational effort for deriving the block-cut graph is $O(|V| + |E|)$.

In contrast to the above definition of the block-cut tree according to [5, 10], we now apply the following simplification: Block-vertices representing blocks that consist of exactly two cut-vertices are redundant in our approach and are therefore removed; a new edge directly connecting the two adjacent cut-vertices is included instead. In Figure 1(b), the block-vertex labeled as "{}" is an example.

Superimposing and Backmapping Augmentation Edges. After the block-cut tree T has been derived from G_0, all augmentation edges in E_a are *superimposed* on T (see, e.g., [7, 16]) as follows: For each edge $\{u, v\} \in E_a$, a corresponding edge $\{u', v'\}$ is created with $u', v' \in V_T$ being the vertices that are associated with u and v, respectively; edge costs are adopted, i.e. $c_{u'v'} = c_{uv}$. From the obtained augmented graph we finally obtain a simple graph, the *block-cut graph* $G_{BC} = (V_T, E_T \cup E_A)$, by deleting self-loops and multiple augmentation edges. In order to derive the original edges $E'_a \subseteq E_a$ corresponding to a solution $E'_A \subseteq E_A$ identified on the block-cut graph, it is necessary to maintain a *back-mapping* from E_A to E_a.

Preprocessing. We can iteratively apply the following basic preprocessing steps to the graph $G_{BC} = (V_T, E_T \cup E_A)$: edge-elimination, fixing of augmentation edges, and shrinking of biconnected components, as described in [20], until we end up with a block-cut tree that needs to be augmented in optimal way.

Connectivity Properties of the Augmented Block-Cut Tree. Before we describe the ILP formulation for augmenting the block-cut tree, we need to study its connectivity properties.

Definition 2. *Let $G_N = (V_N, L)$, $|V_N| \geq 3$, be a connected undirected graph with a given set of vertices $\emptyset \neq C \subseteq V_N$. We say that G_N is* C-vertex-biconnected *if and only if the removal of any single vertex $c \in C$ does not disconnect G_N.*

Denote by $V[P]$ the set of vertices of a path P.

Definition 3. *For two distinct vertices $x, y \in V_N$ and two simple paths P_1 and P_2 connecting them in G_N, we say that P_1 and P_2 are* C-vertex-disjoint *if and only if $(V[P_1] \cap V[P_2] \cap C) = \{x, y\}$.*

The following generalization of Menger's theorem for the C-biconnectivity case holds:

Proposition 1. *An edge-biconnected undirected graph $G_N = (V_N, L)$, $|V_N| \geq 3$, is C-vertex-biconnected if and only if every pair of distinct vertices $x, y \in V_N$ is connected by at least two C-vertex-disjoint paths.*

Our goal will be to augment the block-cut tree at minimum cost so that it becomes V_C-vertex-biconnected.

Definition 4. *A subset $E'_A \subseteq E_A$ is called a* feasible augmentation *of $G_0 = (V, E_0)$, or just* a feasible augmentation, *if after back-mapping of E'_A, we obtain a subset $E'_a \subseteq E_a$ such that the graph $G' = (V, E_0 \cup E'_a)$ is vertex-biconnected.*

The following proposition shows that the problem of augmenting the graph $G_0 = (V, E_0)$ is equivalent to the problem of augmenting the block-cut tree $T = (V_T, E_T)$ using augmentation edges of the block-cut graph E_A.

Proposition 2. *Let $G'_{BC} = (V_T, E_T \cup E'_A)$ be the graph assigned with a subset of augmentation edges $E'_A \subseteq E_A$. Denote by E'_a the set of augmentation edges of G obtained after back-mapping E'_A, and by G' the corresponding augmented graph $G' = (V, E_0 \cup E'_a)$. Then*

1. *G' is vertex-biconnected if and only if G'_{BC} is V_C-vertex-biconnected.*
2. *If E'_A is a feasible augmentation of the block-cut tree T then G'_{BC} is edge-biconnected but not necessarily vertex-biconnected.*

2 Minimum-Cut-Based ILP Formulations

The most effective ILP formulations related to edge/vertex biconnectivity network design problems use an exponential number of constraints and therefore rely on cutting plane and/or branch-and-cut methods. Polyhedral structures related to the biconnectivity property are explored in various papers [6, 9, 22, 26, 31]. It has been recently reported [2, 3] that a BC algorithm has solved instances of the $\{0, 1, 2\}$-survivable network design problem with several thousands of vertices to provable optimality. Hence, the use of the cutting plane framework is a natural approach to solve V2AUG. Our exact algorithm searches for the optimal augmentation of the reduced block-cut tree as described above. In this section we give two minimum-cut-based ILP formulations, an undirected and a directed one, and show that they both provide lower bounds of the same quality.

2.1 An Undirected Minimum-Cut-Based ILP Formulation

Let $E'_A \subseteq E_A$ be a feasible augmentation. According to the Definition 4, every feasible solution of V2AUG can be represented by a characteristic vector x:

$$x_e = \begin{cases} 1, & \text{if } e \in E'_A \\ 0, & \text{otherwise} \end{cases} \qquad \forall e \in E_A.$$

We establish an one-to-one correspondence between the variables x_e and augmentation edges $e \in E_A$ and therefore we use the terms *edge* (graph formulation) and *variable* (integer programming formulation) interchangeably. The same holds for *cutting planes* (or *cuts*) and *inequalities* (or *constraints* or *requirements*). Given a subset of vertices $W \subset V_T$, we denote by $\delta(W)$, $\delta_A(W)$ and $\delta_T(W)$, the edges of the cuts induced by W on $E_A \cup E_T$, E_A and E_T, respectively, i.e.:

$$\delta(W) = \{e = \{i,j\} \in E_A \cup E_T \mid i \in W,\ j \in V_T \setminus W\}, \quad \delta(v) = \delta(\{v\}),$$

$$\delta_A(W) = \delta(W) \cap E_A, \quad \text{and} \quad \delta_T(W) = \delta(W) \cap E_T.$$

Given a single vertex $v \in V_T$, we abbreviate $\delta(\{v\})$ by $\delta(v)$. Further abbreviations are $\delta_{A-v}(W)$ for $\delta_A(W) \setminus \delta(v)$, $\delta_{T-v}(W)$ for $\delta_T(W) \setminus \delta(v)$ and $x(D)$ for $\sum_{e \in D} x_e, D \subseteq E_A$.

Following Proposition 2, an edge set E'_A that augments the block-cut tree T represents a valid vertex-biconnected solution if and only if all cut-vertices in T are covered. This leads to the following ILP formulation of the problem:

$$\text{UCut:} \quad \min \quad \sum_{e \in E_A} c_e \cdot x_e \tag{2}$$

$$x(\delta_{A-v}(W)) \geq 1 \qquad \forall v \in V_C \quad \forall W :\ \emptyset \neq W \subset V_T - v,\ \delta_{T-v}(W) = \emptyset \tag{3}$$

$$x_e \in \{0,1\} \qquad \forall e \in E_A \tag{4}$$

Inequalities (3) are called the *cut-vertex-connectivity* requirements—they ensure V_C-vertex-biconnectivity of the augmented graph $G'_{BC} = (V_T, E_T \cup E'_A)$. In other words, these constraints ensure that a removal of any cut-vertex $v \in V_C$ leaves the augmented graph $G'_{BC} - v = (V_T \setminus \{v\}, (E_T \cup E'_A) \setminus \delta(v))$ connected.

The following lemma is a corollary of Proposition 2.

Lemma 1. *An optimal solution for* UCut *gives an optimal solution to the corresponding V2AUG problem.*

We will make use of the *edge-connectivity constraints* that ensure edge-biconnectivity of the augmented block-cut graph:

$$x(\delta_A(W)) \geq 1, \quad \forall W :\ \emptyset \neq W \subset V_T,\ |\delta_T(W)| = 1. \tag{5}$$

These inequalities are redundant for the ILP formulation (see Proposition 1) and even for the LP-relaxation (see Lemma 2). However, our computational study shows that by separating these constraints in the first step, and asking for V_C-vertex-biconnectivity of G_{BC} in the second step, we can significantly speed up the computation.

In the following, let $\mathcal{P}_{UCut}$ be the polytope corresponding to the UCUT LP-relaxation, i.e., $\mathcal{P}_{UCut}$ contains all points feasible for UCUT disregarding the integer properties of variables:

$$\mathcal{P}_{UCut} = \{x \in [0,1]^{|E_A|} \mid x \text{ satisfies (3)}\}. \tag{6}$$

Lemma 2. *The edge-connectivity inequalities (5) are induced by the cut-vertex-connectivity inequalities (3), i.e. they do not strengthen the LP-relaxation of* UCUT.

Proof. Let x' be a feasible solution of the LP-relaxation of UCUT, i.e., $x' \in \mathcal{P}_{UCut}$. Assume that x' does not satisfy (5), i.e. that there is a subset $\emptyset \neq W' \subset V_T$ such that

$$x'(\delta_A(W')) < 1 \text{ and } |\delta_T(W')| = 1. \tag{7}$$

Denote by $e' = \{u', v'\} \in E_T$ the edge e' such that $\delta_T(W') = \{e'\}$. By construction of the block-cut tree T, at least one of the end-vertices u' and v' is a cut-vertex, say v'. In that case, $\delta_{T-v'}(W') = \emptyset$ and inequality (3) holds, i.e. :

$$x(\delta_A(W')) \geq x(\delta_{A-v'}(W')) \geq 1,$$

which is a contradiction to (7). □

2.2 A Directed Minimum-Cut-Based ILP Formulation

Chimani et al. [3] have shown that, for several vertex-biconnected Steiner network problems, directed minimum-cut-based ILP formulations provide stronger lower bounds than their undirected counterparts. The formulations rely on certain orientation properties of an undirected biconnected graph. An *orientation* of an undirected graph $G_N = (V_N, L)$ is a directed graph $G'_N = (V_N, A_L)$ obtained by uniquely directing each edge from G_N. Robbins [30] has shown that for any graph G, there exists an orientation of G_N with the following property: for every pair of vertices that is edge-biconnected in G_N, there exist two directed paths $(u \mapsto v)$ and $(v \mapsto u)$ in G'_N.

Chimani et al. [2, 3] gave a new orientation-based characterization of vertex-biconnected graphs:

Theorem 1. *[Chimani et al., 2008] An undirected graph $G_N = (V_N, L)$ is vertex-biconnected if and only if for an arbitrarily chosen root $r \in V_N$ there exists an orientation of G_N with* $\text{outdeg}(r) = 1$ *such that, for each $v \in V_N, v \neq r$, there are two directed paths $(r \mapsto v)$ and $(v \mapsto r)$ that are vertex-disjoint except for r and v.*

We now show that the following extension of Theorem 1 holds:

Theorem 2. *[Extension of Theorem 1 to C-biconnectivity] Given an undirected edge-biconnected graph $G_N = (V_N, L)$, with a set of vertices $\emptyset \neq C \subseteq V_N$, the following statements are equivalent:*

1. *G_N is C-vertex-biconnected.*
2. *For an arbitrarily chosen root $r \in C$ there exists an orientation of G_N with* $\text{outdeg}(r) = 1$ *such that, for each $v \in V_N, v \neq r$, there are two directed paths $(r \mapsto v)$ and $(v \mapsto r)$ that are C-vertex-disjoint except for r and v.*
3. *For an arbitrarily chosen root $r \in V_N \setminus C$ there exists an orientation of G_N such that, for each $v \in V_N, v \neq r$, there are two directed paths $(r \mapsto v)$ and $(v \mapsto r)$ that are C-vertex-disjoint except for r and v.*

Proof. 1. $\Rightarrow$ 2.: Assume that G_N is C-vertex-biconnected and that r is an arbitrary vertex from C. We now show how to orient G_N appropriately. At the first step, the maximal biconnected component B_r containing r is detected. Note that there is only one such component, because $r \in C$ and G_N is C-vertex-biconnected. Using the result of Theorem 1, the vertices of B_r are labeled as *visited* and B_r is oriented so that $\text{outdeg}(r) = 1$ and for each $v \in B_r, v \neq r$, there are two directed paths $(r \mapsto v)$ and $(v \mapsto r)$ that are (C-) vertex-disjoint except for r and v. Since G_N is only edge-biconnected, there might exist several non-oriented vertex-biconnected components in G_N. An arbitrary block B is called an *augmenting block* iff exactly one of its vertices is labeled as visited. Obviously, in every iteration there exists at least one augmenting block, and, since the graph is edge-biconnected, there are no trivial blocks. We can orient every augmenting block, starting from its labeled vertex, say v_l, as a root. Obviously, $v_l \notin C$, since it is a cut-vertex of G_N. By orienting the augmenting block B starting with v_l as a root, we ensure that for every vertex $a \in B$, there are vertex-disjoint paths $(v_l \mapsto a)$ and $(a \mapsto v_l)$ such that $\text{outdeg}(v_l) = 1$. Assuming that B and B_r share the common cut-vertex v_l, for any $a \in B$ we get two oriented paths $(r \mapsto a)$ and $(a \mapsto r)$ that are C-vertex-disjoint and obtained by concatenating paths $(r \mapsto v_l) + (v_l \mapsto a)$ and $(a \mapsto v_l) + (v_l \mapsto r)$, respectively. Repeating this procedure, we construct the appropriate orientation without changing the degree of r.

2. $\Rightarrow$ 1.: Assume that there exists a cut-vertex $v_c \in C$ and that there are at least two non-trivial blocks B_1 and B_2 containing v_c. Now take the appropriate orientation for v_c set as a root. Since out-degree of v_c is equal to one, assume that this outgoing arc leads towards B_1. But then, for any vertex $v \in B_2, v \neq r$, there will be no directed path $(r \mapsto v)$, which is a contradiction.

1. $\Rightarrow$ 3.: We choose a root $r \in V_N \setminus C$, label r as visited, and iteratively orient all augmenting blocks in G_N. It is obvious that the obtained orientation ensures the existence of C-vertex-disjoint directed paths $(r \mapsto v)$ and $(v \mapsto r)$ for all $v \in V_N, v \neq r$. Thereby, the out-degree of r will be equal to the number of blocks r is adjacent to.

3. $\Rightarrow$ 1.: Assume that G_N is not C-vertex-biconnected. It is easy to see that there will be at least one block B_1 with maybe several cut-vertices, but exactly one cut-vertex $c \in C$. The cut-vertex c is adjacent to at least one more non-trivial block B_2, $B_1 \neq B_2$. Take the vertex $v_n \in B_1 \cap (V_N \setminus C)$ (observe that there always exists such vertex, since G_N is edge-biconnected, i.e. B_1 is non-trivial). Consider the appropriate orientation for v_n chosen as root. This orientation ensures the existence of two directed C-vertex disjoint paths between v_n and any vertex $v \in B_2$, which is a contradiction with c being a cut-vertex in G_N.

□

In order to define a directed minimum-cut-based ILP formulation for V2AUG, we first transform the block-cut graph G_{BC} into the bidirected graph $\bar{G}_{BC}$ with arc sets A_T and A, corresponding to E_T and E_A, respectively, with arc weights $c_{ij} = c_{ji} = c_e, \forall e = \{i, j\} \in E_A$. The problem of searching for the corresponding feasible augmentation of the block-cut tree $G'_{BC} = (V_T, E_T \cup E'_A)$ at minimum cost, can then be reformulated as the problem of searching for a feasible orientation of the bidirected block-cut graph $\bar{G}'_{BC} = (V_T, A_T \cup A')$, $A' \subseteq A$, at minimum cost.

Since the feasible augmentation leads to a block-cut graph which is edge-biconnected, but not necessarily vertex-biconnected, there might be several vertex-biconnected components to which a block-vertex $v_b \in V_T$ is adjacent to. Hence, by choosing an arbitrary block-vertex v_b to be the root r, we do not need to take care of the corresponding out-degree constraint as in Theorem 1.

As a corollary of Theorem 2 and Proposition 2, for an augmented graph $G' = (V, E_0 \cup E'_a)$, and the corresponding augmented block-cut tree $G'_{BC} = (V_T, E_T \cup E'_A)$, the following theorem holds:

Theorem 3. *Graph G' is vertex-biconnected if and only if for an arbitrarily chosen root $r \in V_B$ there exists an orientation of G'_{BC} such that, for each $v \in V_T, v \neq r$, there are two directed paths $(r \mapsto v)$ and $(v \mapsto r)$ in G'_{BC} that are V_C-vertex-disjoint except for r and v.*

Proof. Assume that G' is vertex biconnected. According to Proposition 2, it follows that the corresponding augmented block-cut graph G'_{BC} is V_C-vertex-biconnected. But then, using Theorem 2, we can find the orientation of G'_{BC} with desired properties.

Conversely, if for any $r \in V_B$ there exists such an orientation, according to Theorem 2, the graph G'_{BC} is vertex-biconnected with respect to $V_T \setminus V_B = V_C$, which further implies that G' is vertex-biconnected. □

To model the solution $\bar{G}'_{BC}$ we use variables $z_{vw} \in \{0,1\}$, $\forall (v,w) \in A_T \cup A$. Without loss of generality, we choose r to be a leaf of T. Furthermore, for all S, $\emptyset \neq S \subset V_T$ we use the following notation:

$$\delta^-(S) = \{(i,j) \in A_T \cup A \mid i \notin S, j \in S\} \quad \text{and} \quad \delta^+(S) = \{(i,j) \in A_T \cup A \mid i \in S, j \notin S\},$$

and for all $v \in V_T$, and $\forall S, \emptyset \neq S \subset V_T \setminus \{v\}$ we define:

$$\delta^-_{\bar{G}_{BC}-v}(S) = \delta^-(S) \setminus (\delta^+(v) \cup \delta^-(v)) \quad \text{and} \quad \delta^+_{\bar{G}_{BC}-v}(S) = \delta^+(S) \setminus (\delta^+(v) \cup \delta^-(v)).$$

For any $D \subseteq A_T \cup A$, let $z(D) = \sum_{ij \in D} z_{ij}$. The ILP formulation based on directed cuts [3] reads then as follows:

$$\text{DCut}: \qquad \min \sum_{(i,j) \in A} c_{ij} \cdot z_{ij} \tag{8}$$

$$z_{ij} + z_{ji} = 1 \qquad \forall \{i,j\} \in E_T \tag{9}$$

$$z_{ij} + z_{ji} \leq 1 \qquad \forall \{i,j\} \in E_A \tag{10}$$

$$z(\delta^-(S)) \geq 1 \qquad \forall S: \emptyset \neq S \subseteq V_T \setminus \{r\} \tag{11}$$

$$z(\delta^+(S)) \geq 1 \qquad \forall S: \emptyset \neq S \subseteq V_T \setminus \{r\} \tag{12}$$

$$z(\delta^-_{\bar{G}_{BC}-v}(S_1)) + z(\delta^+_{\bar{G}_{BC}-v}(S_2)) \geq 1 \qquad \forall v \in V_C, \forall S_1, S_2: \emptyset \neq S_1, S_2 \subseteq V_T \setminus \{r, v\} \tag{13}$$

$$z_{ij} \in \{0,1\} \qquad \forall (i,j) \in A_T \cup A \tag{14}$$

Equalities (9) ensure that arcs of the block-cut tree are included in the solution, whereas with (10) we model directed augmentation arcs. With inequalities (11) and (12) we ensure the edge-biconnectivity of the solution, i.e., that there are directed paths $r \mapsto v$ and $v \mapsto r$, respectively, for any vertex $v \in V_T$. Finally, cut-vertex-disjointness requirements (13) ensure V_C-vertex-disjointness of these two paths, for every $v \in V_T \setminus \{r\}$.

Lemma 3. *An optimal solution for* DCut *gives an optimal solution for the corresponding V2AUG problem.*

We now show that the ILP model based on undirected cuts is equally strong as the directed one. For that purpose, let us define:

$$\mathcal{P}_{DCut} = \{z \in [0,1]^{|A_T \cup A|} \mid z \text{ satisfies (9)-(13)}\}.$$

Theorem 4. DCut *and* UCut *formulations are equally strong, i.e.:*

$$\text{Proj}_x(\mathcal{P}_{DCut}) = \mathcal{P}_{UCut},$$

where $\text{Proj}_x(\mathcal{P}_{DCut}) = \{x \in [0,1]^{|E_A|} \mid z \in \mathcal{P}_{DCut},\ x_{ij} = z_{ij} + z_{ji},\ \forall\{i,j\} \in E_A\}$.

Proof. We prove the equality by showing the mutual inclusion:

$\underline{\text{Proj}_x(\mathcal{P}_{DCut}) \subseteq \mathcal{P}_{UCut}}$**:** Obviously, every directed LP-solution z' is projected into the undirected solution x' satisfying conditions of $\mathcal{P}_{UCut}$.

$\underline{\mathcal{P}_{UCut} \subseteq \text{Proj}_x(\mathcal{P}_{DCut})}$**:** Consider a solution $x' \in \mathcal{P}_{UCut}$ and set $z'_{ij} = z'_{ji} = \frac{1}{2}x'_{ij}$, $\{i,j\} \in E_A$, and $z'_{ij} = z'_{ji} = \frac{1}{2}$, $\{i,j\} \in E_T$. Assume now that the so constructed solution z' violates one of the edge-biconnectivity constraints (11) (or equivalently (12)). Then, there exists a set $S' \subset V \setminus \{r\}$ such that $z'(\delta^-(S')) < 1$. From Lemma 2 and (5) we know that $\frac{1}{2}x'(\delta(S')) = z'(\delta^-(S')) \geq 1$ which leads to a contradiction.

Assume finally that there exists a cut-vertex $v' \in V_C$ and two subsets $S'_1, S'_2 \subseteq V_T \setminus \{r, v'\}$ such that z' violates vertex-biconnectivity constraints (13), i.e. such that:

$$z'(\delta^-_{\bar{G}_{BC}-v'}(S'_1)) + z'(\delta^+_{\bar{G}_{BC}-v'}(S'_2)) < 1. \tag{15}$$

If $S'_1 = S'_2$ or $S'_1 = V_T \setminus S'_2$, we immediately get a contradiction with undirected cut-vertex-disjointness constraints (3). Therefore, assume that $S'_1 \neq S'_2$ and $S'_1 \neq V_T \setminus S'_2$. Since E_T is a spanning tree, for every set $S \subset V_T$, we know that $|\delta(S) \cap E_T| \geq 1$. Obviously, there must exist two edges $e_1 = \{v_1, w_1\} \in \delta(S'_1) \cap E_T$ and and $e_2 = \{v_2, w_2\} \in \delta(S'_2) \cap E_T$ such that $e_1 \neq e_2$. The assumption $v' \notin \{v_1, w_1, v_2, w_2\}$ automatically leads to a contradiction, because then $e_1 \in \delta_{G_{BC}-v'}(S'_1) \cap E_T$, and $e_2 \in \delta_{G_{BC}-v'}(S'_2) \cap E_T$, and due to (3):

$$z'(\delta^-_{\bar{G}_{BC}-v'}(S'_1)) \geq z'(\delta^-_{\bar{G}_{BC}-v'}(S'_1) \cap A_T) \geq \frac{1}{2}, \tag{16}$$

and

$$z'(\delta^+_{\bar{G}_{BC}-v'}(S'_2)) \geq z'(\delta^+_{\bar{G}_{BC}-v'}(S'_2) \cap A_T) \geq \frac{1}{2}. \tag{17}$$

So let $v' = v_1$. Observe that after removing v' from $\bar{G}_{BC}$ (and G_{BC}), (17) still holds. If $|\delta_{G_{BC}-v'}(S'_1) \cap E_T| = 0$, due to (3) we have:

$$z'(\delta^-_{\bar{G}_{BC}-v'}(S'_1)) = \frac{1}{2}x'(\delta_{G_{BC}-v'}(S'_1)) \geq \frac{1}{2}.$$

If, otherwise, $|\delta_{G_{BC}-v'}(S_1') \cap E_T| \geq 1$, then (16) holds too, which finally concludes the proof.

□

Theorem 4 confirms our choice of taking the undirected cut-based model as a basis for the development of the BCP algorithm: UCut has less variables and less constraints, thereby providing lower bounds of the same quality as DCut.

We have to point out that one can derive a compact ILP formulation for V2AUG based on multi-commodity flows on directed graphs, called DFlow, in a similar way as in [3]. However, since the formulations DCut and DFlow are equally strong in general [3], it follows that DFlow cannot improve the quality of lower bounds of UCut either. Furthermore, we know that flow-based formulations are computationally inferior to cut-based models (see [3]), which is an additional argument for modelling V2AUG with UCut.

3 A Branch-and-Cut-and-Price Algorithm for V2AUG

Within this section we first propose a branch-and-cut (BC) algorithm for UCut based on the classical separation of cut-vertex- and edge-connectivity-inequalities. We then show how to extend the BC approach with a column generation method. To initialize upper bounds we use values derived from a metaheuristic framework [20]. An efficient LP-rounding heuristic extended with a local improvement method is presented, as well.

3.1 Ingredients of the Branch-and-Cut Approach

We developed and implemented a branch-and-cut algorithm based on our UCut formulation. For a general description of the branch-and-cut scheme see, e.g., [23]. Since we can solve the separation problem in polynomial time (see below), it follows that we can also solve the underlying LP-problem in polynomial time [23]. Hence, despite exponentially many UCut-constraints, we can obtain an optimal fractional solution of the LP-relaxation in polynomial time at the root node of the branch-and-bound tree.

We now describe the specific ingredients of our BC algorithm.

Initialization. We obtain an adequate choice of the initial set of constraints by choosing the *degree inequalities* that correspond to the leaves of the block-cut tree (according to the requirements

(5) for $|W| = 1$):

$$x(\delta_A(v)) \geq 1, \forall v \in V_T,\ |\delta_T(v)| = 1.$$

For the optimal LP-solution obtained in such a way, further connectivity constraints may be introduced as cutting planes described below.

Separation. Given a solution to the current partial LP, we build the support graph $G_x = (V_T, E_T \cup E_A, c')$, whose edge weights are defined as:

$$c'_e = \begin{cases} 1, & \text{if } e \in E_T \\ x'_e, & \text{otherwise} \end{cases},$$

where x'_e represents the fractional value of the corresponding variable in the current LP. In each iteration, violated constraints are detected by means of minimum weight cuts in the support graph. For the computation of minimum cuts, we use an efficient algorithm proposed by Padberg and Rinaldi [24] and implemented by Jünger et al. [15]. The details of this implementation and an exhaustive comparison of a variety of minimum weight cut algorithms are given in [15].

We perform separation in two stages. As the block-cut graph needs to be *edge-biconnected*, we first impose the *edge-connectivity requirements*. A violated edge-connectivity constraint is detected if the minimum cut of weight less than two is found. In the second separation phase, when all edge-connectivity constraints are satisfied, we check if there are some uncovered cut-vertices. Therefore, for each cut-vertex $v \in V_C$, we reduce the support graph G_x by eliminating v from it. In other words, we search for the minimum weight cut in the graph $(V_T \setminus \{v\}, E_T \cup E_A \setminus \delta(v))$. If the cut we found is less than one, the corresponding constraint is inserted into the system.

In both phases, before resolving the LP, we add *multiple disjoint connectivity cuts* (see, e.g., [21]). For this purpose, edge weights of the detected cut are set to one in the support graph, and a new minimum cut is calculated. Our computational experiments have confirmed that this strategy has advantage in comparison to the separation of single connectivity cuts, which is mainly due to the fact that adding several violated cuts at once saves the time-consuming process of solving several LPs. Finally, our preliminary experiments have also shown that it is beneficial to insert all violated constraints corresponding to the uncovered cut-vertices before resolving the LP.

Branching and Enumeration Rules. We branch on a single variable according to `CloseHalfExpensive` strategy [32]. Suppose x is a fractional solution of the currently solved linear program. Among the set of variables "close" to 0.5, we select one with the maximum absolute cost, i.e. with the maximum objective value coefficient (note that our edge weights are

positive). The *best first search* strategy has been used as the default enumeration strategy: from the set of open subproblems the "most promising" one is selected. In our case, the node with the maximal local lower bound is said to be the most promising one.

Initializing Upper Bounds. Using good upper bounds plays an important role in the design of branch-and-bound based algorithms. The better the upper bound, the more nodes in the branch-and-bound tree can be fathomed. For the initialization of upper bounds, we used the memetic algorithm (MA) proposed in [20] which is the state-of-the-art approach to V2AUG. When solving small and medium-sized instances, we observed that there is a trade-off between the MA's running time and the running time needed to prove optimality. Thus, by default, we used a *weaker termination criterium*: the population size was set to 100 and MA was terminated after a new *best* solution was not found during the last $\Omega = 1000$ iterations. These parameters are set according to preliminary tests. Because of MA's non-deterministic nature, we ran it with a fixed seed value. We will refer to this initialization strategy as the *weak initialization of upper bounds*.

3.2 The Branch-and-Cut-and-Price Algorithm

In this section we propose an enhancement of the previous BC approach, the *branch-and-cut-and-price method*, which is achieved by embedding a column generation method into each node of the branch-and-cut tree. For an introduction to the column generation approach embedded in an enumeration framework (e.g., branch-and-price, branch-and-cut-and-price), see, e.g., [4]. We also extend BC with a local improvement algorithm as a *primal heuristic*. In the following, we highlight the column generation procedure and the primal heuristic.

Sparse and Reserve Graph Pricing. The size of feasible solutions in our problem is bounded by n (the number of vertices), while the number of variables is bounded by $\binom{n}{2} - n + 1$ in the worst case, when the graph is complete and G_0 is a tree. It has been shown that column generation may also be used to speed-up the computation of (integer) linear programs even when the number of variables is polynomial in input size, if the size of any basic solution is comparatively sparse [14, 4]. This increase of speed is achieved by using the *sparse and reserve graph pricing technique*.

The small set of active variables used to initialize the restricted master problem corresponds to a subgraph of the original graph which is called the *sparse graph* (see [8, 25, 29], for example). In addition to the sparse graph, Jünger et al. [14] proposed the usage of the *reserve graph*. In a

single pricing iteration, all edges from the reserve graph with negative reduced costs are added to the restricted problem and the LP is resolved. If all edges from the reserve graph price out correctly, the *complete pricing* is performed: from the set of all inactive edges, those with negative reduced costs are determined and added to the LP at once. As proposed in [14] in the context of TSP, we use the 5-nearest neighbor graph to initialize the sparse graph, and we set the reserve graph to be its difference to the 10-nearest neighbor graph.

During pricing, we also fix some inactive variables by their reduced costs [25, 32]. If our current branch-and-cut node is the root of the remaining branch-and-cut tree, we search for inactive variables $x_e, e \in E_A$ such that $c(LB) + r_e > UB_g$, where $c(LB)$ denotes the last computed lower bound, UB_g represents the global upper bound, and r_e represents the reduced cost of the variable x_e. Such variables x_e can be discarded forever. The other variables are inserted in the list of possible candidates that can be priced in later iterations.

Primal Heuristic. The initial upper bound of the branch-and-cut algorithm proposed so far can be improved only if the LP-solution is integer feasible, which happens rather rarely. Therefore we enhance the method with a *primal heuristic* which is applied at each node of the branch-and-bound tree, before a branching step is applied, in order to generate new and better feasible solutions.

The fractional LP-solutions occurring in the lower bound computations may give a good insight into the structure of optimum or near optimum feasible solutions. Our heuristic is designed in the following way: Starting from the fractional solution x' of the last LP, we generate a support block-cut graph using the edges of the block-cut tree T and the edges (i.e. variables) from the set E_{prHeur}, where

$$E_{prHeur} = \{e \in E_A \mid x'_e \geq p_{prHeur}\}.$$

$p_{prHeur} \in [0, 1]$ is a fractional parameter which controls the influence of the LP solution on the generation of feasible solutions.

If the support graph is not biconnected, we first make it feasible by adding an additional subset of augmentation edges, as described in the following. Later, we apply a local improvement procedure that removes redundant edges from the support graph.

Repairing the Support Graph: Iteratively, non-redundant edges are randomly selected from $E_A \setminus E_{prHeur}$ and included in the support graph. This process is repeated until all cut-vertices are covered. Intuitively, cheaper edges appear in optimum solutions more likely than

expensive edges. Therefore, the selection of edges for inclusion is biased toward cheaper edges [20, 27].

Local Improvement: Finally, from such a feasible solution, say E'_A, we eliminate the redundant edges by using the local improvement described below, with one exception; to better exploit the LP-solution, we forbid elimination of edges with $x_e = 1$.

A feasible augmentation E'_A is said to be *locally optimal* with respect to the number of edges, if the removal of any edge $e \in E'_A$ violates the biconnectivity property of graph $G' = (V, E_0 \cup E'_a)$, where $E'_a \subseteq E_a$ is the set of original augmentation edges corresponding to E'_A. The local improvement operator makes a given feasible solution locally optimal by removing redundant edges. An edge $e \in E'_a$ is said to be redundant if its removal does not violate the biconnectivity property of G'. As a first step, the algorithm identifies *obviously essential edges* that must remain in E'_a, i.e. those edges that are the only possibility to connect a certain cut-component of a cut-vertex v_c to any other of v_c's cut-components. The remaining not *obviously essential edges* from E'_A, are then processed one-by-one in decreasing-cost order. They are temporarily removed from E'_A, and the cut-vertices to which covering e contributes are checked. If any of them are now uncovered, e is not redundant and therefore included in E_A once more.

In the worst case, the total computational effort of this local improvement procedure is $O(|E_A|^2 \cdot |V_T| \cdot \alpha(|V_T|, |E_A|))$ per call.

4 Computational Results

In this section, we analyze the performance of the proposed branch-and-cut-and-price approach. In particular, first we investigate impacts of extending the BC algorithm with pricing. Then we study the role of the primal heuristic. Finally, we examine the trade-off between initializing BCP with high-quality upper bounds and its overall performance.

Our algorithm is implemented using C++ under Linux and all the experiments reported in this paper have been run on a Pentium IV/2.8 GHz PC with 2GB RAM. The only exception are the results reported in Section 4.1, where, for comparison purposes, we used a Pentium III/800 MHz machine.

We used the ABACUS 2.3 software system [13] as a generic implementation of the branch-and-cut-and-price approach [4, 32]. CPLEX 7.1 [12] has been used as the LP-solver. In order to make a fair comparison to the existing approaches, the preprocessing proposed in [20] has been applied

to all considered instances, and to *all approaches* (including metaheuristics and approximation algorithms) mentioned within the paper. If not otherwise mentioned, the reported running time includes the preprocessing time as well.

4.1 Benchmark Instances

As already observed, shrinking reduces the problem of augmenting a general connected graph G_0 to the problem of augmenting a tree. Therefore, we consider only instances such that the fixed graph G_0 is a spanning tree. We considered two types of graphs that we refer to as: (1) randomly generated graphs and (2) real-world graphs derived from the TSPLIB. These benchmark instances are designed to capture the behavior of BCP on different types of networks.

Randomly Generated Graphs. Table 1 shows the characteristics of 810 instances originally proposed in [20], divided in 27 groups `A1` to `R2`, each consisting of 30 graphs. We call instances `A1-D4` *KRZ instances*, since they were created using a random generator proposed by Khuller et al. [17, 33]. KRZ considered graphs with up to 50 vertices and proposed 3 different density types (sparse, medium and dense) defined by the following functions, respectively: $f_1(n) = 3n$, $f_2(n) = n\ln(n)$, $f_3(n) = \binom{n}{2}$. For all unordered pairs of vertices, given a density type $i \in \{1, 2, 3\}$, a random integer value r is drawn uniformly in $\{1, \ldots, n(n-1)\}$. If $r < f_i(n)$, an edge is created with the integer cost uniformly distributed in $\{1, \ldots, \binom{n}{2}\}$. A random spanning tree is then determined on the graph, yielding the set of fixed edges E_0. When creating graphs with $|V| \geq 70$ and for density type $i = 1$, we were not able to generate vertex-biconnected graphs by applying those rules, and therefore, we changed the function f_1 to $f_1'(n) = 0.3\binom{n}{2}$. Instances are grouped according to their complexity as far as approaches tested in [20] are considered. The density types are provided in the column denoted by "type".

Instances of groups `M,N` and `R` are generated using primarily the same ideas, but with fixed density values (0.15, 0.25 and 0.5, respectively) and with edge costs taken from several intervals as provided in Table 1. Column *dens* provides the density of each group, whereas column "$c_e \in$" gives the intervals from which the edge costs are drawn.

Table 1 also shows the comparison between our branch-and-cut algorithm, described in Section 3.1, and the memetic algorithm (MA) proposed in [20]. The results are averaged over 30 different instances of the same group. Column *OPT* represents the averaged optimal values. The next two columns provide MA results: the average percentage gap (%-*gap*) and the average running time in seconds (t [s]). The last four columns are devoted to the branch-and-cut algorithm:

Table 1. Characteristics of randomly generated instances and average results of the memetic algorithm and the BC approach. The branch-and-cut algorithm solved all the instances to optimality in less than a minute.

						MA		BC			
Group	$\|V\|$	*dens*	$c_e \in$	type	*OPT*	%-*gap*	t [s]	t [s]	SP	Levels	LPs
A1	20	0.16	[1..190]	1	511.50	0.00	0.00	0.00	1.17	1.07	1.37
A2	30	0.10	[1..435]	1	1764.77	0.00	0.00	0.00	1.07	1.03	1.33
A3	40	0.08	[1..780]	1	4055.47	0.00	0.01	0.01	1.00	1.00	1.10
A4	30	0.12	[1..435]	2	1948.07	0.00	0.01	0.01	1.27	1.13	1.47
A5	40	0.10	[1..780]	2	3753.87	0.00	0.02	0.01	1.27	1.13	1.47
B1	60	0.05	[1..1770]	1	13426.03	0.00	0.03	0.02	1.10	1.03	1.37
B2	20	0.50	[1..190]	3	163.77	0.00	0.12	0.00	1.60	1.30	2.27
B3	50	0.06	[1..1225]	1	8311.93	0.00	0.02	0.02	1.13	1.07	1.30
B4	50	0.08	[1..1225]	2	7131.37	0.00	0.08	0.02	1.53	1.27	2.00
B5	60	0.07	[1..1770]	2	12460.57	0.00	0.12	0.03	1.27	1.13	1.70
B6	70	0.06	[1..2415]	2	19849.73	0.00	0.32	0.05	1.13	1.07	1.47
C1	80	0.06	[1..3160]	2	27085.03	0.00	0.41	0.07	1.27	1.13	1.57
C2	90	0.05	[1..4005]	2	40478.83	0.00	0.49	0.09	1.13	1.07	1.33
C3	100	0.05	[1..4950]	2	52441.30	0.00	0.62	0.12	1.07	1.03	1.33
C4	30	0.50	[1..435]	3	341.50	0.00	0.38	0.01	1.07	1.03	1.53
D1	70	0.15	[1..2415]	1′	7339.93	0.00	1.68	0.37	1.27	1.13	1.50
D2	40	0.50	[1..780]	3	762.70	0.00	0.75	0.05	1.33	1.17	1.90
D3	90	0.15	[1..4005]	1′	12773.33	0.00	3.91	1.64	1.27	1.13	1.57
D4	80	0.15	[1..3160]	1′	9886.33	0.00	2.87	1.19	1.27	1.13	1.43
D5	100	0.15	[1..4950]	1′	13489.10	0.02	5.90	2.13	1.40	1.20	1.80
M1	70	0.15	[10..1000]	-	3492.33	0.00	1.70	0.46	1.40	1.20	2.00
M2	80	0.15	[10..1000]	-	3266.33	0.00	2.86	1.04	1.27	1.13	1.77
M3	90	0.15	[10..1000]	-	3433.33	0.00	4.31	1.62	1.13	1.07	1.37
N1	100	0.25	[11..50]	-	389.93	0.17	9.50	2.83	19.07	3.50	18.77
N2	110	0.25	[11..50]	-	413.63	0.39	13.72	2.95	7.00	2.53	7.63
R1	200	0.50	[1..100]	-	128.93	0.08	39.78	19.31	2.20	1.50	2.47
R2	200	0.50	[5..100]	-	331.54	0.42	58.52	16.84	3.60	1.63	3.50

the average running time in seconds (t [s]), the average number of generated subproblems (*SP*), the average number of generated levels in the branch-and-cut tree (*Levels*), and the average number of solved linear programs (LPs). To be able to compare our BC approach to MA, we performed the experiments on the same machine: a Pentium-III/800 MHz.

All instances of this group are now solved to provable optimality. The results show that the branch-and-cut algorithm is significantly faster than MA, which is the fastest heuristic approach so far. According to results reported in [20], the algorithm of Khuller et al. [17], which is the best approximation algorithm with factor 2, did not terminate for the instances with 200 vertices (`R`-group) within the allowed time of 20 000 seconds on the same machine. In the branch-and-cut algorithm, all instances (with the exception of the ones of the `N`- and `R`-group) are solved using slightly more than one subproblem on average. This means that, in most cases, the cutting plane method performed at the root node solves the underlying problem to optimality. Furthermore, this indicates that the lower bounds obtained by relaxing our ILP formulation are quite strong for these types of graphs. For all these reasons, we consider these randomly generated graphs as easy, and conduct our study on significantly larger and more challenging real-world networks described below.

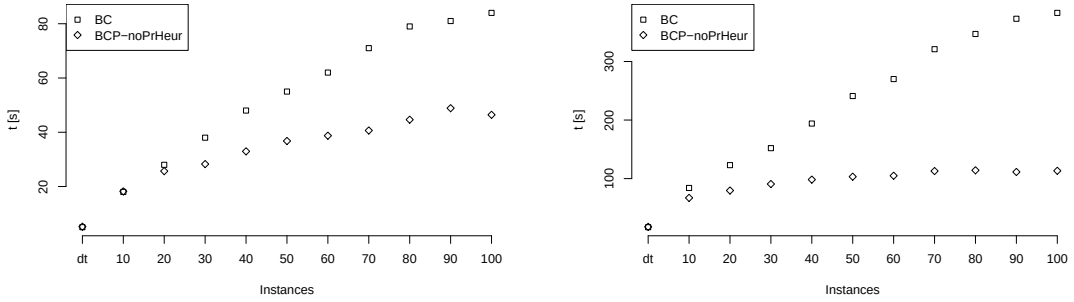
Instances derived from Reinelt's TSP-library (TSPLIB). In order to check the practical feasibility of the proposed BCP algorithm, we considered several instances randomly selected from the TSPLIB [28]. In the selected graphs defining *traveling salesman problems*, vertices are defined as *cities* or *drilling points*. Due to the real-world aspect of these instances, we take them as the basis for the rest of our computational study. Instances `pr226`, `lin318`, `pr439`, and `pcb442` are of Euclidean type, meaning that the vertices represent points in the Euclidean plane and edge costs are the Euclidean distances of the corresponding points rounded up to the nearest integer value. The instance `pa561` is a complete graph with edge costs directly given by a matrix – the triangle inequalities are satisfied.

Since all these instances represent complete graphs, and as the incomplete graphs are of particular interest as well, additional sparse instances `pr226-sp`, `lin318-sp`, `pr439-sp`, `pcb442-sp`, and `pa561-sp` are derived from the original TSPLIB-graphs as follows. For each vertex of the original graph, we take the edges to its $\lceil |V| \cdot 10\% \rceil$ nearest neighbors, i.e. we create the 10%-*nearest-neighbor graphs*. In the case of instance `pr226-sp`, the 10%-nearest-neighbor graph turned out not to be biconnected, and the 15%-nearest-neighbor graph is used instead. For the Euclidean instances we further calculate the Delaunay triangulation yielding additional sparse instances `pr226-dt`, `lin318-dt`, `pr439-dt`, and `pcb442-dt`. We consider graphs in which k% of nearest

neighbors of each vertex are included in G, where $k \in \{20, 30, \ldots, 90\}$. Finally, we consider two additional groups with a larger number of vertices: `d1291` and `d2103`, derived from the TSPLIB as well. In all TSPLIB-derived graphs, the fixed graph G_0 is set to be the minimum spanning tree.

4.2 Branch-and-Cut vs. Branch-and-Cut-and-Price

We first test the benefits of incorporating column generation, based on sparse and reserve graph techniques, into the BC framework. The default initialization of the sparse and reserve graph, with 5%− and 10%− nearest-neighbor graphs (NNGs) has been changed in the case of `pr226` and `pr439` groups. In order to ensure feasibility, we initialize the sparse graph with 8%− and 6%− NNGs, and the reserve graph with 12%− and 10%−NNGs, respectively. For all computational experiments presented in this section we used a Pentium IV/2.8GHz with 2 GB RAM. Except for pricing, all other attributes of BC and BCP are kept the same.



(a) `pr439` (b) `pcb442`

Fig. 2. Comparing the running time (in seconds) of the branch-and-cut algorithm (BC) and the branch-and-cut-and-price algorithm without the primal heuristic (BCP-noPrHeur). Per instance group, results for 11 instances with different densities of $k\%$ (x-axis) are shown.

To be able to investigate the role of pricing in improving the BC performance, we first switch off the primal heuristic. The results depicted on Figure 2 show the comparison of the running time between the branch-and-cut algorithm as described in Section 3.1 and the BCP algorithm without the primal heuristic (BCP-noPrHeur). Furthermore, we subtracted the preprocessing time (see [20]), thus comparing only the computational effort of "pure" exact algorithms. The obtained results show that the incorporation of pricing into the branch-and-cut framework for

V2AUG significantly speeds up the computation. For the `pcb442` (`pr439`) group, the overall running time can be reduced by about 50% (35%) on average. In addition, one observes that the density of an instance does not substantially influence the running time, if pricing is used. Conversely, for dense graphs, the overall running time without pricing may be up to three times longer (see, for example, `pcb442` compared to `pcb442-sp`).

4.3 Benefits of the Primal Heuristic

Figure 3 depicts one example that shows the advantage of using the primal heuristic. The performance of the BCP algorithm is better, if only promising variables (i.e. those whose LP-values are greater than a certain threshold value p_{prHeur}) are used within the upper bounding procedure. We observe that, if all possible augmentation edges (i.e. $x_{ij} \geq 0$) are used within the upper bounding procedure, there is no significant difference between the performance of the BCP algorithm with or without the primal heuristic.

Further experiments have shown that the most robust performance is obtained by setting p_{prHeur} to a standard value of 0.5, usually used within similar upper bounding procedures. Indeed, setting the value of p_{prHeur} too high, often leads to infeasible solutions that need to be repaired by randomly adding additional augmentation edges. This leads to solutions of worse quality and to the higher computational time needed to repair them.

4.4 Influence of Upper Bounds

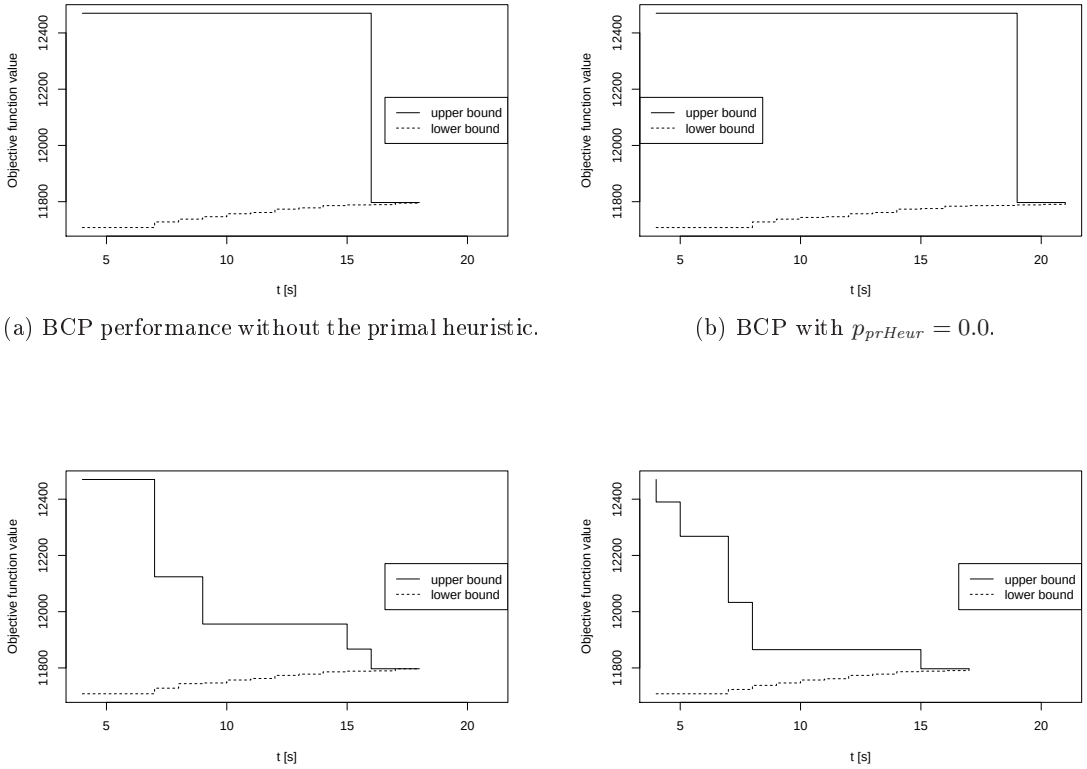
We now investigate the influence of the initialization with high-quality upper bounds to the overall performance of the BCP algorithm. As our heuristic choice, we used the memetic algorithm [20]. We considered the following two initialization settings:

- In the default BC / BCP implementation, as described in Sections 3.1 and 3.2, respectively, we used the weak initialization. The population size was set to 100 and each MA run was terminated when no new best solution could be identified during the last $\Omega = 1\,000$ iterations. This setting corresponds to a fast initialization, and therefore to low-quality upper bounds.
- In BCP-Strong-Init, the convergence criteria for MA was stronger (the same as described in [20]), thus $\Omega = 10\,000$ while the population size was set to 800. Here, we allow longer running time to obtain high-quality feasible solutions.

Table 2 summarizes the results for all TSPLIB-derived instances that could be solved to provable optimality. For the three strategies mentioned above, the running time in seconds (t [s]),

Table 2. Comparing the BC algorithm and the BCP algorithm (both with the weak initialization) with the BCP algorithm with the strong initialization (BCP-Strong-Init). The best running time is highlighted.

Instance	OPT		BC			BCP				BCP-Strong-Init			
		t_{prep} [s]	t [s]	SP	LP	t [s]	SP	LP	t_{best} [s]	t [s]	SP	LP	t_{best} [s]
pr226-dt	25152	0.3	**0.9**	5	9	1.0	5	9	0	3.2	5	9	2
pr226-sp	22824	0.4	**2.0**	3	5	2.2	5	10	1	8.9	5	10	8
pr226 (20)	22824	0.5	**2.3**	3	4	**2.3**	3	9	2	8.9	3	9	8
pr226 (30)	22824	0.7	**3.7**	5	11	3.9	7	18	3	10.1	3	14	9
pr226 (40)	22824	0.9	3.9	5	9	**3.5**	5	13	3	11.9	5	12	10
pr226 (50)	22824	1.3	4.5	5	11	**3.6**	5	14	3	12.4	7	14	10
pr226 (60)	22824	2.0	5.0	5	11	**4.4**	5	13	4	12.1	7	16	11
pr226 (70)	22824	2.5	5.8	5	12	**5.3**	5	13	4	12.4	5	19	11
pr226 (80)	22824	2.6	6.1	7	12	**5.0**	5	10	4	12.1	5	13	11
pr226 (90)	22824	2.7	6.7	5	10	**5.4**	3	13	5	13.4	5	11	12
pr226	22824	2.6	6.5	5	10	**5.2**	3	11	5	14.0	5	12	13
pr226-avg		1.5	4.3	4.8	9.5	**3.8**	4.6	12.1	3.1	10.8	5.0	12.6	9.5
lin318-dt	12013	2.4	**1.3**	3	5	1.5	3	6	1	7.2	3	6	7
lin318-sp	11797	2.5	13.1	73	53	**11.9**	59	66	10	20.5	59	65	19
lin318 (20)	11797	3.2	17.7	73	53	**12.4**	59	65	11	19.6	59	65	18
lin318 (30)	11797	6.3	21.4	73	53	**13.2**	59	66	11	22.9	59	65	21
lin318 (40)	11797	10.1	24.5	73	53	**14.0**	59	65	12	29.6	59	65	28
lin318 (50)	11797	10.6	30.6	81	56	**14.5**	59	65	13	24.8	59	65	23
lin318 (60)	11797	19.9	32.7	73	53	**16.3**	59	66	14	28.3	59	65	26
lin318 (70)	11797	24.1	36.4	73	53	**16.8**	59	66	15	30.0	59	65	28
lin318 (80)	11797	29.7	39.5	73	53	**18.1**	59	68	16	24.6	59	65	23
lin318 (90)	11797	32.5	45.8	85	62	**18.3**	59	68	16	28.3	59	65	26
lin318	11797	23.1	44.6	81	56	**18.2**	59	68	16	28.5	59	65	26
lin318-avg		15.0	28.0	69.2	50.0	**14.1**	53.9	60.8	12.3	24.0	53.9	59.6	22.3
pr439-dt	28310	8.5	**5.1**	15	19	5.4	15	19	1	13.7	15	19	9
pr439-sp	26800	10.0	18.8	41	37	**14.8**	27	31	13	25.3	27	31	24
pr439 (20)	26800	14.1	28.8	45	38	**19.2**	31	32	17	38.8	31	32	37
pr439 (30)	26800	32.7	38.6	45	41	**21.9**	31	33	19	50.2	31	34	48
pr439 (40)	26800	32.0	48.8	45	37	**25.7**	31	32	23	54.6	31	34	51
pr439 (50)	26800	49.5	55.4	45	41	**29.8**	31	33	27	57.9	31	35	55
pr439 (60)	26800	76.0	62.9	45	39	**31.4**	31	35	28	58.0	31	35	55
pr439 (70)	26800	89.7	71.8	45	40	**32.8**	31	34	30	60.9	31	32	58
pr439 (80)	26800	99.7	79.0	45	41	**36.4**	31	36	33	65.9	31	32	63
pr439 (90)	26800	107.9	81.8	45	40	**40.7**	31	33	37	69.4	31	33	66
pr439	26800	73.3	84.8	45	39	**40.1**	31	35	37	69.6	31	35	66
pr439-avg		53.9	52.3	41.9	37.5	**27.1**	29.2	32.1	24.1	51.3	29.2	32.0	48.4
pcb442-dt	10328	60.7	**17.5**	97	99	18.1	97	101	18	28.4	89	100	28
pcb442-sp	10460	12.0	84.1	253	195	**62.1**	195	201	58	82.0	195	202	78
pcb442 (20)	10460	18.6	123.2	253	195	86.2	253	198	80	**81.8**	195	201	77
pcb442 (30)	10460	34.5	152.6	253	195	**83.5**	195	201	76	96.1	195	201	90
pcb442 (40)	10460	36.1	195.0	253	195	**92.1**	195	201	82	105.3	195	200	96
pcb442 (50)	10460	46.4	241.1	253	191	**96.5**	195	200	86	112.0	195	199	102
pcb442 (60)	10460	78.8	270.4	253	195	**99.6**	195	201	89	118.9	195	200	109
pcb442 (70)	10460	93.1	321.7	253	195	**118.7**	253	198	107	131.6	195	201	121
pcb442 (80)	10460	101.4	347.4	253	195	**104.7**	195	201	94	120.5	195	201	110
pcb442 (90)	10460	108.0	373.7	253	195	**107.7**	199	196	96	132.2	195	201	122
pcb442	10460	74.0	383.5	253	191	**108.4**	199	196	96	125.8	199	196	114
pcb442-avg		60.3	228.2	238.8	185.5	**88.9**	197.4	190.4	80.2	103.1	185.7	191.1	95.2



(a) BCP performance without the primal heuristic.

(b) BCP with $p_{prHeur} = 0.0$.

(c) BCP with $p_{prHeur} = 0.2$.

(d) BCP with $p_{prHeur} = 0.5$.

Fig. 3. Gap versus time plot for `lin318 (70)` instance. Due to the application of the primal heuristic, good feasible solutions can be found earlier.

the total number of generated subproblems (SP) and the total number of solved LPs (LP) are given. Additionally, for BCP and BCP-Strong-Init, t_{best} $[s]$ shows the time when the optimal solution has been detected. Column t_{prep} $[s]$ gives the number of seconds we needed to run the preprocessing [20]. The results indicate unambiguously that the pricing together with the primal heuristic substantially improves the performance of the branch-and-cut algorithm. The overall running time can be reduced by more than 50% on average (the average/median BC running time is 78/34, whereas the corresponding BCP time is 33/18 seconds). Also the size of the branch-and-bound tree can be reduced by about 20% on average (the average/median number of SPs needed to solve BC is 88/45, whereas the corresponding BCP number is 71/31). Conversely, comparing the running time of BCP and BCP-Strong-Init, we observe that the initialization with the high-quality upper bounds slows down the overall performance. As we will see below, the real benefits of the strong initialization are first visible when the instances become much larger.

Table 3 shows the results for large graphs for which we were not able to find optimal solutions. For the groups `pa561`, `d1291` and `d2103`, our branch-and-cut-and-price algorithm terminated prematurely because of memory limitations. The only exception was the graph `d2103 (2)`, for which we found the optimum. Therefore, we measured the following optimality gap:

$$gap_g = \frac{UB_g - LB_g}{LB_g} \times 100\%,$$

where UB_g represents the cost of the best known feasible augmentation (obtained either within MA, or inside of BCP), and LB_g is the global lower bound. The *optimality gap* (gap_g) expresses that the solution with cost UB_g is at most gap_g% more expensive than the optimal solution.

In Table 3, we consider three different settings: The default BCP implementation as described in Section 3.2 (BCP), the BCP algorithm without the primal heuristic (BCP-noPrHeur), and BCP with the strong initialization (BCP-Strong-Init). In the UB and UB_{full} columns, we show the upper bounds obtained after the weak and the strong initialization, respectively. For each of these settings, we provide the total running time in seconds (t [s], including preprocessing and MA running time) and the optimality gap (gap_g).

Table 3 indicates that for large instances it is recommendable to run MA to obtain as good solutions as possible ($\Omega = 10\,000$), and to keep higher diversity (population size of 800). While for small and medium-size instances the computation of high-quality upper bounds can slow-down the optimization, for larger instances it helps significantly in reducing the gap between the global lower bound and the best-known feasible solution. One observes that the BCP-Strong-Init approach produces optimality gaps of up to 2%, the gap of the BCP approach is below 4%, while the BCP algorithm without the primal heuristic and with the weak initialization produces the worst solutions, with optimality gaps of up to 14%.

5 Conclusions

This paper represents the first theoretical and computational study on exact approaches to the edge-weighted vertex biconnectivity augmentation problem. Using the connectivity properties of the block-cut graph, we have proposed a new orientation-based characterization of augmented block-cut trees. We have derived two ILP formulations with an exponential number of inequalities relying on connectivity properties of the block-cut graph. In contrast to recent results published by Chimani et al. [3] for solving related vertex-biconnected Steiner network problems, we have shown that the ILP model on undirected graphs is as strong as the model on directed graphs, and is therefore preferable in practice.

Table 3. Comparing three BCP settings for large instances: BCP without the primal heuristic (BCP-noPrHeur), BCP with the primal heuristic (BCP), and BCP with the primal heuristic and the strong initialization (BCP-Strong-Init). Both BCP-noPrHeur and BCP use the weak initialization of upper bounds (see Section 3.1). The best optimality gap values are highlighted.

Instance	UB	BCP-noPrHeur		BCP		UB_{full}	BCP-Strong-Init	
		t [s]	gap_g	t [s]	gap_g		t [s]	gap_g
pa561-sp	794	580.8	2.7	596.1	1.9	784	593.2	**1.4**
pa561 (20)	828	682.3	7.1	681.5	1.8	786	694.0	**1.6**
pa561 (30)	849	750.5	9.8	754.8	3.3	782	763.8	**1.1**
pa561 (40)	837	934.6	8.2	933.6	3.4	785	774.1	**1.5**
pa561 (50)	851	1094.4	10.0	1067.1	3.2	785	852.2	**1.4**
pa561 (60)	805	1018.4	4.1	1208.7	3.2	784	929.0	**1.4**
pa561 (70)	872	1771.4	12.8	1377.7	3.1	785	938.8	**1.6**
pa561 (80)	879	2226.3	13.7	1401.5	3.4	788	1031.2	**2.0**
pa561 (90)	844	2243.6	9.1	1380.0	2.7	785	1042.0	**1.6**
pa561	878	1754.3	13.5	1690.4	3.4	786	842.6	**1.7**
d1291 (2)	12298	1443.4	5.4	1450.7	1.5	11788	1493.2	**1.0**
d1291 (5)	12210	1571.5	4.6	1583.4	**1.5**	11855	1624.6	**1.5**
d1291 (10)	12634	1772.0	8.3	1768.2	**1.7**	11924	1810.7	**1.7**
d1291 (30)	13357	3499.8	14.7	3305.1	**1.9**	11997	3385.3	**1.9**
d2103 (2)	7633	3766.6	**0.0**	3723.5	**0.0**	7490	3749.7	**0.0**
d2103 (5)	7610	4637.4	2.7	4653.9	**0.4**	7521	4692.1	**0.4**
d2103 (10)	7691	5478.1	3.9	5468.6	**0.7**	7503	5529.3	**0.7**

To solve V2AUG, we have extended the traditional branch-and-cut algorithm by a column generation strategy. Furthermore, we have studied the role of the primal heuristic and the importance of the initialization with good upper bounds for the overall BCP performance.

We have obtained optimal solutions for complete graphs with more than 400 vertices. For instances with more than 2000 vertices, we have achieved optimality gaps that are strictly below 2%.

Acknowledgements

The author is greatly indebted to: Michael Jünger for providing the implementation of the minimum-cut algorithm [15] and the framework for the sparse and reserve graph pricing [32]; to Markus Chimani, Maria Kandyba, Petra Mutzel and Günther Raidl for very useful discussions on this and closely related topics.

References

1. J. Bang-Jensen, M. Chiarandini, and P. Morling, A computational investigation of heuristic algorithms for 2-edge-connectivity augmentation, Networks (2009), to appear.
2. M. Chimani, M. Kandyba, I. Ljubić, and P. Mutzel, Strong formulations for 2-node-connected Steiner network problems, Proceedings of 2nd Annual International Conference on Combinatorial Optimization and Applications, Lecture Notes in Computer Science, Vol. 5165, Springer, 2008, pp. 190–200.
3. M. Chimani, M. Kandyba, I. Ljubić, and P. Mutzel, Orientation-based models for $\{0, 1, 2\}$-survivable network design: Theory and practice, Mathematical Programming (2009), to appear.
4. M. Elf, C. Gutwenger, M. Jünger, and G. Rinaldi, "Branch-and-cut algorithms for combinatorial optimization and their implementation in ABACUS", Computational Combinatorial Optimization, M. Jünger and D. Naddef (Editors), Lecture Notes in Computer Science, Vol. 2241, Springer, 2001, pp. 157–222.
5. K. P. Eswaran and R. E. Tarjan, Augmentation problems, SIAM Journal on Computing 5 (1976), 653–665.
6. B. Fortz and M. Labbé, Exact and heuristic algorithms for the design of survivable networks with bounded rings, Mathematical Programming 93 (2002), 27–54.
7. G. N. Frederickson and J. Jájá, Approximation algorithms for several graph augmentation problems, SIAM Journal on Computing 10 (1981), 270–283.
8. M. Grötschel and O. Holland, Solving matching problems with linear programming, Mathematical Programming 33 (1985), 243–259.

9. M. Grötschel, C. Monma, and M. Stoer, Polyhedral and computational investigations for designing communication networks with high survivability requirements, Operations Research 43 (1995), 1012–1024.
10. F. Harary, Graph Theory, Addison-Wesley, Reading, MA, 1969.
11. T.-S. Hsu, Simpler and faster biconnectivity augmentation, Journal of Algorithms 45 (2002), 55–71.
12. ILOG CPLEX, webpage. `http://www.ilog.com/products/cplex/`.
13. M. Jünger, ABACUS – A Branch-And-CUt System, webpage. `http://www.informatik.uni-koeln.de/abacus/`.
14. M. Jünger, G. Reinelt, and S. Thienel, Provably good solutions for the traveling salesman problem, Zeitschrift für Operations Research 40 (1994), 183–217.
15. M. Jünger, G. Reinelt, and S. Thienel, Practical performance of efficient minimum cut algorithms, Algorithmica 26 (2000), 172–195.
16. S. Kersting, G. R. Raidl, and I. Ljubić, A memetic algorithm for vertex-biconnectivity augmentation, Applications of Evolutionary Computing, Proceedings of EvoWorkshops 2002, Lecture Notes in Computer Science, Vol. 2279, Springer, 2002, pp. 102–111.
17. S. Khuller, B. Raghavachari, and A. Zhu, A uniform framework for approximating weighted connectivity problems, Proc 10th Ann ACM-SIAM Symp on Discrete Algorithms, 1999, pp. 937–938.
18. S. Khuller and R. Thurimella, Approximation algorithms for graph augmentation, Journal of Algorithms 14 (1993), 214–225.
19. I. Ljubić and J. Kratica, A genetic algorithm for the biconnectivity augmentation problem, Proceedings of the 2000 Congress on Evolutionary Computation, IEEE Press, 2000, pp. 89–96.
20. I. Ljubić and G. R. Raidl, A memetic algorithm for minimum-cost vertex-biconnectivity augmentation of graphs, Journal of Heuristics 9 (2003), 401–427.
21. I. Ljubić, R. Weiskircher, U. Pferschy, G. Klau, P. Mutzel, and M. Fischetti, An algorithmic framework for the exact solution of the prize-collecting steiner tree problem, Mathematical Programming 105 (2006), 427–449.
22. T. L. Magnanti and S. Raghavan, Strong formulations for network design problems with connectivity requirements, Networks 45 (2005), 61–79.
23. G. L. Nemhauser and L. A. Wolsey, Integer and combinatorial optimization, John Wiley & Sons, New York, 1998.
24. M. Padberg and G. Rinaldi, An efficient algorithm for the minimum capacity cut problem, Mathematical Programming 47 (1990), 19–36.
25. M. Padberg and G. Rinaldi, A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems, SIAM Review 33 (1991), 60–100.
26. S. Raghavan, Formulations and algorithms for network design problems with connectivity requirements, Ph.D. Thesis, Massachusetts Institute of Technology, 1995.
27. G. R. Raidl, An efficient evolutionary algorithm for the degree-constrained minimum spanning tree problem, Proceedings of the 2000 Congress on Evolutionary Computation, IEEE Press, 2000, pp. 104–111.

28. G. Reinelt, TSPLIB, webpage. http://elib.zib.de/pub/mp-testdata/tsp/tsplib/tsplib.html.
29. G. Reinelt, Fast heuristics for large geometric traveling salesman problems, ORSA Journal on Computing 4 (1992), 206–217.
30. H. Robbins, A theorem on graphs with an application to a problem of traffic control, American Mathematical Monthly 46 (1939), 281–283.
31. M. Stoer, Design of survivable networks, Vol. 1531, Lecture Notes in Mathematics, Springer, 1992.
32. S. Thienel, A Branch-And-CUt System, Ph.D. Thesis, University of Cologne, Germany, 1995.
33. A. Zhu, A uniform framework for approximating weighted connectivity problems, B.Sc. Thesis, University of Maryland, MD, 1999.