TECHNISCHE UNIVERSITÄT WIEN
Institut für Computergraphik und Algorithmen

A Memetic Algorithm for Minimum-Cost Vertex-Biconnectivity Augmentation of Graphs

I. Ljubić, G. R. Raidl

Forschungsbericht / Technical Report

TR-186-1-02-01

June 2002

Favoritenstraße 9-11 / E186, A-1040 Wien, Austria
Tel. +43 (1) 58801-18601, Fax +43 (1) 58801-18699
www.cg.tuwien.ac.at

A Memetic Algorithm for Minimum-Cost Vertex-Biconnectivity Augmentation of Graphs

Ivana Ljubić and Günther R. Raidl*

Institute of Computer Graphics and Algorithms,
Vienna University of Technology,
Favoritenstraße 9–11/186, 1040 Vienna, Austria

{ljubic|raidl}@ads.tuwien.ac.at

Contact author: Günther Raidl

phone: +43(1)58801-18616, fax: +43(1)58801-18699

REVISED VERSION

December 2002

Abstract

This paper considers the problem of augmenting a given graph by a cheapest possible set of additional edges in order to make the graph vertex-biconnected. A real-world instance of this problem is the enhancement of an already established computer network to become robust against single node failures. The presented memetic algorithm includes effective preprocessing of problem data and a fast local improvement strategy which is applied before a solution is included into the population. In this way, the memetic algorithm's population consists always of only feasible, locally optimal solution candidates. Empirical results on two sets of test instances indicate the superiority of the new approach over two previous heuristics and an earlier genetic algorithm.

Keywords:
vertex-biconnectivity, connectivity augmentation, network survivability, memetic algorithm, evolutionary computation

There are two main aspects for robust communication networks: reliability and survivability. Reliability is the probability that a network functions according to a specification. Survivability is the ability of a network to perform according to a specification after some failure. In many applications it is not acceptable that the failure of a single service node—be it a computer, router, or other device—leads to a disconnection of other nodes. Survivability is extremely important in modern telecommunication networks, in particular in backbones. Redundant connections need to be established to provide alternative routes in case of a temporary break of any one node.

This kind of robustness of a network is in graph theory described by means of *vertex-connectivity*. A network is said to be vertex k-connected if at least k nodes must be deleted (together with the set of their incident links) in order to separate it into two or more disconnected components. A k-connected network, $k \geq 2$, is said to be survivable. The probability of a second failure before a first one is repaired is often neglected; k-connected networks with $k \geq 3$ are usually considered not worth the additional costs. Therefore, this paper focuses on the most common case of biconnected networks, where $k = 2$.

In the vertex-biconnectivity augmentation problem, a connected but not vertex-biconnected network is given. Thus, there are some critical nodes, called *cut-points*, whose removal would separate the network into unconnected components. We say that we *cover a cut-point* when we add some links to ensure that the removal of this node no longer disconnects the network. The global aim is to identify a set of additional links with minimum total costs in order to cover all cut-points.

Formally, the *vertex-biconnectivity augmentation problem for graphs* (V2AUG) is defined as follows. Let $G = (V, E)$ be a vertex-biconnected, undirected graph with node set V and edge set E representing all possible connections. Each edge $e \in E$ has associated $cost(e) > 0$. A connected, spanning, but not vertex-biconnected subgraph $G_0 = (V, E_0)$ with $E_0 \subset E$ represents a fixed, existing network, and $E_a = E \setminus E_0$ is the set of edges that may be used for augmentation. The objective is to determine a subset of these candidate edges $E_s \subseteq E_a$ so that the augmented graph $G_s = (V, E_0 \cup E_s)$ is vertex-biconnected and

$$cost(E_s) = \sum_{e \in E_s} cost(e) \tag{1}$$

is a minimum. See Fig. 2a for an example.

Eswaran and Tarjan (1976) have shown this problem to be NP-hard. Exact optimization algorithms, like branch-and-bound or cutting-plane approaches, have been developed for V2AUG but are limited

in their applicability to instances of moderate size. For large instances, effective heuristic methods are needed.

This article describes a new meta-heuristic approach, whose basic structure is outlined in Fig. 1. From the problem's original graphs, a more compact block-cut graph G_A is deterministically derived. Then, a new enhanced preprocessing is applied, which may shrink the block-cut graph substantially by fixing or discarding certain augmentation edges in safe ways. Some further data structures allowing the following optimization to be implemented in a more efficient way are also created during preprocessing. The core of the whole system is a new evolutionary algorithm that uses problem specific variation operators and strongly interacts with a local improvement procedure—a so-called *memetic algorithm* (MA) (Moscato, 1999). This MA searches for a low-cost solution on the reduced block-cut graph. The best solution found is finally mapped back to a solution for the original V2AUG instance.

In the sequel, Sect. 1 introduces the block-cut graph. An overview on former approaches to V2AUG and related problems is given in Sect. 2. Section 3 explains the preprocessing in detail, and Sect. 4 focuses on the memetic algorithm. In Sect. 5, empirical results are presented and compared to those of three previous heuristic approaches, including a genetic algorithm. Conclusions are drawn in Sect. 6.

1 The Block-Cut Graph

All maximal subgraphs of the fixed graph G_0 that are already vertex-biconnected, i.e. the vertex-biconnected components, are referred to as *blocks*. Any two blocks share at most a single node, and this node is a cut-point; its removal would disconnect G_0 into at least two components.

A *block-cut tree* $T = (V_T, E_T)$ with node set V_T and edge set E_T is an undirected tree that reflects the relations between blocks and cut-points of a fixed graph G_0 in a simpler way (Eswaran and Tarjan, 1976). Figure 2b illustrates this. Two types of nodes form V_T: cut-nodes and block-nodes. Each cut-point in G_0 is represented by a corresponding cut-node in V_T, each maximal vertex-biconnected block in G_0 by a unique block-node in V_T.

A cut-node $v_c \in V_T$ and a block-node $v_b \in V_T$ are connected by an undirected edge (v_c, v_b) in E_T if and only if the cut-point corresponding to v_c in G_0 is part of the block represented by v_b. Thus, cut-nodes and block-nodes always alternate on any path in T. The resulting structure is always a

tree, since a cycle would form a larger vertex-biconnected component, and thus, the block-nodes would not represent maximal biconnected components.

A block-node is associated with all nodes of the represented block in G_0 excluding cut-points. If the represented block consists of cut-points only, the block-node is not associated with any node from V. In this way, each node from V is associated with exactly one node from V_T, but not vice-versa.

In contrast to the previous definition of the block-cut tree according to Eswaran and Tarjan (1976), we apply here the following simplification: Block-nodes representing blocks that consist of exactly two cut-points only are unimportant and therefore removed; a new edge directly connecting the two adjacent cut-nodes is included instead. In Fig. 2b, the block-node labeled "{}" is an example.

After the block-cut tree T has been derived for graph G_0, all augmentation edges in E_a are *superimposed* on T forming a new edge-set E_A: For each edge $(u, v) \in E_a$, a corresponding edge (u', v') is created with $u', v' \in V_T$ being the nodes that are associated with u, respectively v; edge costs are adopted, i.e., $cost((u', v')) = cost((u, v))$. The so-called *(augmented) block-cut graph* $G_A = (V_T, E_T \cup E_A)$ may be a multi-graph containing self-loops and multiple edges between two nodes. However, applying the following safe reductions yields a simple graph:

1. Self-loops $(u, u) \in E_A$, as, e.g., edge e'_1 in Fig. 2b, are discarded. They can never help in establishing biconnectivity.

2. Each augmentation edge that connects the same nodes as an edge from E_T is discarded, since such an edge can also never help in establishing biconnectivity. See edge e'_2 in Fig. 2b.

3. Augmentation edges connecting two cut-nodes that are adjacent to the same block-node in T are also discarded because of the same reason; see edge e'_3 in Fig. 2b.

4. From multiple augmentation edges connecting the same nodes from V_T, only one with minimum weight is retained; see edge e'_4 in Fig. 2b when assuming $cost(e'_4) < cost(e'_5)$. The more expensive edges may never appear in an optimum solution.

In order to be finally able to derive the original edges $E_s \subseteq E_a$ corresponding to a solution $S \subseteq E_A$ identified on the block-cut graph, it is necessary to maintain a back-mapping from E_A to E_a.

The computational effort for deriving the block-cut graph is linear in the number of edges of the original graph G ($O(|E|)$), since all maximal biconnected subgraphs can be found in this time and each edge needs to be considered only once.

2 Previous Work

Eswaran and Tarjan (1976) were the first investigating V2AUG and showed it to be NP-hard. An exact polynomial-time algorithm could only be found for the special case when G is complete and each edge has unit costs (Hsu and Ramachandran, 1993).

Frederickson and Jájá (1981) provided an approximation algorithm for the general case which finds a solution within a factor 2 of the optimum, supposing graph G_0 is connected. If G_0 is not necessarily connected, the approximation factor increases to 3, however, we do not consider this case here. The algorithm includes a preprocessing step that transforms the fixed graph G_0 into the corresponding block-cut tree, superimposes the augmentation edges, and performs the basic reductions as described in the last section. The augmented block-cut graph is further extended to a complete graph such that there is an augmentation edge for each pair of nodes $(u, v) \notin E_T$. All these augmentation edges get new "reduced" costs according to the following definition and maintain back-references to the original augmentation edges where the costs come from:

$$cost'(u, v) = \min\left(\{cost(x, y) \mid (x, y) \in E_A \wedge u, v \text{ are on the } (x, y)\text{-path in } T\} \cup \{\infty\}\right) . \qquad (2)$$

In the main part of the algorithm, the block-cut tree T is directed toward an arbitrarily chosen leaf r, the root, yielding an ingoing arborescence. Each directed tree-edge is assigned zero costs. Each cut-node is substituted by a star-shaped structure including new dummy-nodes in order to guarantee that strongly connecting the block-cut tree implies vertex-biconnectivity of the underlying fixed graph G_0. Two different types of augmentation edges are distinguished: A *back-edge* connects a node with one of its descendants in the directed block-cut tree; all other augmentation edges are called *cross-edges*. Back-edges are directed from the node nearer to the root toward the node farther away; cross-edges are replaced by pairs of reversely directed edges.

A *minimum outgoing spanning arborescence* (MOSA) of a weighted directed graph with a fixed root r is a directed spanning tree of minimum weight such that all nodes except r have incoming degree one, and no edge is directed toward r. Frederickson and Jájá derive such a MOSA for the directed block-cut graph to obtain the solution's edge-set E_s. When using an efficient MOSA-algorithm as described by Gabow et al. (1986), the total computational effort is $O(|V|^2)$. Frederickson and Jájá (1982) further point out some relations between V2AUG and the traveling salesman problem.

This approximation algorithm has been improved by Khuller and Thurimella (1993). The main difference is that the extension of the block-cut graph to a complete graph is omitted. Instead,

each cross-edge (u, v) is replaced by two reversely directed cross-edges (u, v) and (v, u) and two back-edges $(\mathrm{lca}(u, v), u)$ and $(\mathrm{lca}(u, v), v)$, where $\mathrm{lca}(u, v)$ denotes the least common ancestor of u and v in T, i.e. the first node the paths from u to r and v to r have in common. Back-edges are again directed from the node nearer to the root toward the node farther away. Further, no dummy nodes are included, but each augmentation edge (v_c, v) going out of some cut-point v_c is replaced by an edge (v_b, v), where v_b is the node adjacent to v_c on the undirected path from v_c to v in T. The algorithm exhibits a time complexity of only $O(|E| + |V| \log |V|)$, but has still the approximation factor 2. Practical results of this algorithm can be found in the empirical comparison in Sect. 5.

An iterative approach based on Khuller and Thurimella's algorithm has been proposed by Zhu et al. (1999); for more details, see also Zhu (1999). In each step, a *drop*-heuristic measures the gain of each augmentation edge if it would be included in a final solution. This is achieved by calling the MOSA-algorithm for each edge once with its cost set to zero and once with its original cost. The edge with the highest gain is then fixed, and its cost are permanently set to zero. The process is repeated until the obtained MOSA has zero total costs. Furthermore, the whole algorithm is applied with each leaf of the block-cut tree becoming once the root, and the overall cheapest solution is the final one. Although the theoretical approximation factor remains 2, practical results are usually much better than when applying Khuller and Thurimella's algorithm; our empirical comparison in Sect. 5 also supports this. However, time requirements are raised substantially.

A straight-forward genetic algorithm for V2AUG has been proposed by Ljubić and Kratica (2000). This algorithm is based on a binary encoding in which each bit corresponds to an edge in E_a. Standard uniform crossover and bit-flip mutation are applied. Infeasible solutions are repaired in Lamarckian way by a greedy algorithm which temporarily removes cut-points one by one and searches for the cheapest augmentation edges that reconnect the separated components. The major disadvantage of this genetic algorithm is its high computational effort, which mainly comes from the repair strategy having a worst-case running time of $O(|V|\,|E_a| \log |V|)$ per candidate solution.

Another, weaker kind of connectivity property is *edge-biconnectivity*. It means that a graph remains connected after the removal of any single edge. While vertex-biconnectivity implies edge-biconnectivity, the reverse does in general not hold. Similar algorithms as for V2AUG have been applied to the edge-biconnectivity augmentation problem (E2AUG). From the algorithmic point-of-view, E2AUG is easier to deal with, since it does not require the block-cut graph data structure.

The works from Eswaran and Tarjan (1976), Frederickson and Jájá (1981), Khuller and Thurimella

(1993), and Zhu et al. (1999) also address E2AUG. Raidl and Ljubić (2002) describe an effective evolutionary algorithm for E2AUG, which scales well to large problem instances and outperforms several previous heuristics. A compact edge set encoding and special initialization and variation operators that include a local improvement heuristic are applied.

Based on this algorithm for E2AUG, the memetic algorithm for V2AUG presented in this article has been developed. Preliminary results were reported in Kersting et al. (2002). Major differences to the evolutionary algorithm for E2AUG lie in the underlying data structures (e.g., the now necessary block-cut graph), the preprocessing, the recombination and mutation operators, the local improvement algorithm, and the way how this local improvement is integrated in the evolutionary algorithm. While it is relatively easy to check and eventually establish the cover of a single fixed edge in case of E2AUG, this is significantly harder to achieve for a cut-node in the V2AUG-case, especially in an efficient way: A critical fixed edge can always be covered by a single augmentation edge, and it is obvious which augmentation edges are able to cover the critical edge. On the other side, a combination of multiple augmentation edges is in general necessary to completely cover a cut-node.

There are further problem classes related to V2AUG and E2AUG:

Finding a minimum-cost edge or vertex k-connected spanning subgraph of a graph (without any fixed edges) is also known to be NP-hard for $k \geq 2$. Approximation algorithms are described in (Khuller, 1997). Cheriyan et al. (2001) developed an improved approximation algorithm for the vertex k-connectivity case and give a survey on former approaches. To our knowledge, meta-heuristics have not yet been applied to this problem.

In the context of graph drawing, Fialgo and Mutzel (1998) developed an algorithm for augmenting graphs that must remain planar.

Another class of related problems is the augmentation of a multi-graph, i.e., a graph that may contain multiple edges between the same vertices, with the smallest number of unweighted edges so that the resulting graph becomes edge or vertex k-connected. In particular the edge k-connectivity case turned out to be an easier problem: Watanabe and Nakamura (1987) described a polynomial time algorithm for solving the problem to optimality. In case of vertex k-connectivity, exact polynomial time algorithms are known for $k \in \{2, 3, 4\}$; whether the problem is NP-hard for general $k \geq 5$ is still an open question. A recent study on this topic can be found in Ishii (2000).

The more general problem of designing a minimum-cost network with individually specified connectivity requirements for each node—the so-called *survivable network design problem*—has been attacked by Stoer (1992) using a polyhedral approach. By means of cutting-plane techniques the algorithm is able to find optimal or near-optimal solutions for instances of small and moderate size. Monma and Shallcross (1989) considered a variant of this problem in which the connectivity requirements of each node are limited to $\{0, 1, 2\}$.

Recently, Fortz (2000) studied a new kind of survivable network design problem with *bounded rings*. It includes an additional constraint limiting the maximum length of cycles for which no shortcuts exist. The author provides a study of the underlying polyhedron and proposes several classes of facet-defining inequalities used in a branch-and-cut algorithm. Several heuristics are also proposed in order to solve real-world instances of larger size.

3 Preprocessing

The memetic algorithm's preprocessing derives the block-cut graph from the fixed graph G_0 and the set E_a of augmentation edges as described in Sect. 1. In addition, some more sophisticated deterministic rules are applied in order to further reduce the block-cut graph, and other supporting data structures needed for an efficient implementation of the main algorithm are created. The following subsections describe these mechanisms in detail.

3.1 When is a Cut-Node Covered?

A block-cut tree's edge $e \in E_T$ is said to be *covered* by an augmentation edge $e_A = (u, v) \in E_A$ if and only if e is part of the unique path in T connecting u with v. In order to completely cover a cut-node $v_c \in V_T$, all its incident tree-edges need to be covered, but this is in general not sufficient.

If v_c and its incident edges are removed from T, the tree falls apart into l connected components $C_1^{v_c}, \ldots, C_l^{v_c}$, where l is the degree of v_c in T; we call them *cut-components* of v_c; Figure 3 illustrates this. To completely cover v_c, at least $l - 1$ augmentation edges are needed such that all cut-components $C_1^{v_c}, \ldots, C_l^{v_c}$ are united into one connected graph.

We say that an augmentation edge $e_A = (u, v) \in E_A$ *contributes in covering the cut-node* v_c, if and only if two tree-edges incident to v_c are covered by e_A. Such an augmentation edge is obviously not incident to v_c and always connects two cut-components $C_i^{v_c}$ and $C_j^{v_c}$.

For any cut-node $v_c \in V_T$, let $\Gamma(v_c) \subseteq E_A$ be the set of augmentation edges that contribute in covering v_c. Furthermore, for each $e_A \in E_A$, let $\Psi(e_A) \subseteq V_T$ be the set of cut-nodes to whose covering e_A contributes, i.e., $\Psi(e_A) = \{v_c \in V_T \mid e_A \in \Gamma(v_c)\}$. Preprocessing explicitly computes and stores the sets $\Gamma(v_c)$ for all cut-nodes and the sets $\Psi(e_A)$ for all augmentation edges as supporting data structures. This is done by first performing a depth-first search on T and storing for each node its depth and a reference to its parent node, in order to be able to efficiently determine the tree-path between any pair of nodes. Then, the computation of all sets $\Gamma(v_c)$ and $\Psi(e_A)$ can be performed in $O(|E_A|\,|V_T|)$ time. The needed space for these data structures is bounded above by $O(|E_A|\,|V_T|)$. In the average case, however, T is a natural and not degenerated tree having diameter $O(\log |V_T|)$. Then, Γ and Ψ need space $O(|E_A| \log |V_T|)$.

For each entry $e \in \Gamma(v_c)$, preprocessing also stores references to the two tree-edges being incident to v_c and covered by e; we denote them by $e_{T1}^{v_c}(e)$ and $e_{T2}^{v_c}(e)$. They directly reflect the two cut components edge e can connect.

In the memetic algorithm, it is necessary to efficiently check if a certain cut-node is covered by a subset of augmentation edges $S \subseteq E_A$. With the precomputed Γ and Ψ and the aid of a temporary union-find data structure with weight balancing and path compression (Aho et al., 1983, pp. 183–189), this check can be performed in nearly linear time $O(|S|)$. In most cases the degree of the cut-node v_c is less than four. Then, even no union-find data structure is needed, since it is sufficient to check whether each of the tree-edges incident to v_c is covered by some augmentation edge being not incident to v_c.

3.2 Reducing the Block-Cut Graph

In addition to the simple reductions of the block-cut graph described in Sect. 1, we apply the following more sophisticated rules which are partly adopted from the preprocessing for E2AUG in Ljubić and Raidl (2001). These rules are safe in the sense that they never prevent the following optimization from finding an optimal solution.

Edge Elimination: If there are two edges $e_A, e'_A \in E_A$, $cost(e_A) \leq cost(e'_A)$, and e_A covers all those tree-edges that are covered by e'_A (in addition to others), e'_A is obsolete and can be discarded; see Fig. 4a. All such edges can be identified in $O(|V_T|^2)$ time as a byproduct of a

dynamic programming algorithm from Frederickson and Jájá (1981) for computing the reduced costs of Eq. (2).

Fixing of Edges: An edge $e_A \in E_A$ must be included in any feasible solution to the V2AUG problem if it represents the only possibility to connect a cut-component $C_i^{v_c}$ of a cut-node v_c to any other cut-component of v_c. In more detail, we consider for each cut-node v_c its set $\Gamma(v_c)$ and look for those edges being the only ones able to cover one of the tree-edges incident to v_c. Such augmentation edges are fixed by moving them from E_A to E_T; see edge e''_A in Figs. 4b and 4c. The corresponding original augmentation edges from E_a are permanently marked to be included in any future solution. The whole procedure runs in $O(|V_T|\,|E_A|)$ time.

Shrinking: By fixing an edge, a cycle is introduced in T. This cycle forms a new vertex-biconnected component that can be shrinked into a single new block-node v^* as shown in Fig. 4d. Let $Z \subseteq V_T$ be the set of nodes forming the cycle. The following rules are applied:

1. Each block-node $v_b \in Z$ is remapped to v^*. All tree-edges between v_b and a node $u \notin Z$ are remapped to (v^*, u). Each cut-node $v_c \in Z$ having degree two is now completely covered and therefore handled in the same way. All the edges connecting nodes in Z are removed.

2. The remaining cut-nodes $v_c \in Z$ are not remapped to v^*. Instead, their membership to the new block is expressed via new edges (v^*, v_c).

3. All augmentation edges incident to one of the nodes in Z are superimposed anew on the modified block-cut tree according to the rules of Sect. 1.

After shrinking all cycles in T, all modifications are also reflected to the supporting data structures. Owing to the reductions, more edges may become available for elimination and/or fixing. Therefore, all reduction steps are repeated until no further shrinking is possible.

An upper bound for the total effort of preprocessing is $O(|V_T|^2|E_A|)$ since edge elimination, the fixing of edges, and shrinking may theoretically iteratively be applied up to $O(V_T)$ times. However, this happens only in extreme situations and the expected total effort is lower, as also the empirical results in Sect. 5 document.

4 The Memetic Algorithm

It is well known that classical evolutionary algorithms are usually less efficient in fine-tuning solutions in complex search-spaces (Michalewicz, 1996). For many hard combinatorial optimization problems combinations of evolutionary algorithms and local improvement techniques have been applied with great success. In a memetic algorithm, candidate solutions created by an evolutionary algorithm framework are fine-tuned by some local improvement procedure. The exploration abilities of the evolutionary algorithm are complemented with the exploitation capabilities of local improvement. For a more detailed introduction to memetic algorithms, see Moscato (1999).

The memetic algorithm this article proposes for V2AUG is based on a straight-forward steady-state evolutionary algorithm as shown in Fig. 5. In each iteration, k-ary tournament selection with replacement (Blickle, 1997) is performed in order to select two parental solutions for mating. A new candidate solution is always created by recombining these parents, mutating it with a certain probability, and applying local improvement. Such a solution replaces always the worst solution in the population with one exception: To guarantee a minimum diversity, a new candidate whose set of augmentation edges S is identical to that of a solution already contained in the population is discarded (Raidl and Gottlieb, 1999).

As a central element of the memetic algorithm, local improvement is applied to each randomly created initial solution and to each solution derived by recombination and possibly mutation. In this way, the evolutionary algorithm's population always contains only locally optimal solutions with respect to the number of augmentation edges. The following subsections describe in detail how solutions are represented and local improvement, initialization, recombination, and mutation are performed.

4.1 Representation of Solutions

Many evolutionary algorithms for combinatorial optimization problems represent candidate solutions by vectors of fixed length and apply classical operators as k-point or uniform crossover and position-wise mutation. Ljubić and Kratica (2000) followed this concept with their genetic algorithm for V2AUG and represented a solution by a vector of $|E_a|$ Booleans indicating which augmentation edges are included in the solution.

The main disadvantage of this approach is that created candidate solutions need not to be feasible; an expensive repair strategy, which also reduces the variation operators' locality and heritability is necessary. Furthermore, the memory effort for storing a solution is $O(|E_a|)$.

In the memetic algorithm, a candidate solution is represented by directly storing references to all the augmentation edges of $S \subseteq E_A$ in the form of a hash-table. In this way, only $O(|S|) = O(|V|)$ space is needed, since $|S| < |V|$ in any solution that is locally optimal with respect to the number of edges (in fact, $|S| \ll |V_T|$ in most larger instances). Using a hash-table allows an edge to be added, deleted, or checked for existence in constant time.

4.2 Local Improvement

A feasible candidate solution S is said to be *locally optimal* with respect to the number of edges, if the removal of any edge $e \in S$ violates the biconnectivity-property of graph $G_s = (V, E_0 \cup E_s)$, where $E_s \subseteq E_a$ is the set of original augmentation edges corresponding to S united with the edges fixed during preprocessing. An edge $e \in S$ is said to be *redundant* if its removal does not violate the biconnectivity-property of G_s. The local improvement operator shown in Fig. 6 and described in the following makes a given feasible solution locally optimal by removing redundant edges. It is specifically designed to perform efficiently on sparse solutions where $|S| = O(|V_T|)$, since the solutions created by initialization, recombination, and mutation do not usually have many redundant edges.

As first step, the algorithm identifies so-called *obviously essential edges* that must remain in S. An edge $e \in S$ is obviously essential if it is the only one from S able to connect a certain cut-component $C^i_{v_c}$ of a cut-node v_c to any other of v_c's cut-components—compare the fixing of edges during preprocessing. Such obviously essential edges from S are determined efficiently by finding each tree-edge e_T incident to a cut-node v_c and covered only once by an edge $e \in S$ that is not incident to v_c; e is then obviously essential. The worst-case time complexity of this part of the algorithm, when implemented as shown in the pseudo-code, is $O(|S|\,|V_T|)$. However, since $|\Psi(e)| = O(\log |V_T|)$ in the expected case, the average running-time is $O(|V_T| + |S| \log |V_T|)$.

The remaining not obviously essential edges from S, in the pseudo-code denoted by set R, are then processed one-by-one in decreasing-costs order. Each edge $e \in R$ is temporarily removed from S, and the cut-nodes in whose covering e contributes, i.e. all $v_c \in \Psi(e)$, are checked if they remain

covered (see Sect. 3.1). If any of them is now uncovered, e is not redundant and therefore included in S again.

In the worst case, the total computational effort of this local improvement procedure is $O(|S|^2|V_T|)$ per call. The example in Fig. 7 illustrates this: Assuming each block-node in the shown block-cut graph represents a single node in the original graph G_0, there are $|V_T| = (|V| - 2)/3 = O(|V|)$ cut-nodes having all degree four. No augmentation edge is obviously essential. $O(|S|) = O(|V_T|)$ augmentation edges incident to block-nodes 1 and 2 contribute in the covering of each cut-node. On the other side, each of these augmentation edges contributes in the covering of $O(|V_T|)$ cut-nodes. Since the time for checking whether a single cut-node remains covered when a certain augmentation edge is removed is $O(|S|)$, it takes $O(|V_T|\,|S|)$ time to completely check an augmentation edge for redundancy, and the overall effort is $O(|S|^2|V_T|) = O(|V_T|^3)$ per solution.

However, since $|\Psi(e)| = O(\log|V_T|)$ on average, the average time for checking one edge from R for redundancy is $O(|S|\log|V_T|)$, and the average total time for one complete local improvement is $O(|V_T| + |S|\log|V_T| + |R|\,|S|\log|V_T|)$. In case of the memetic algorithm's candidate solutions, usually most edges are obviously essential; thus, $|R|$ is generally small.

Another possibility for checking an edge $e \in S$ for redundancy is to temporarily remove it and to check whether the augmented graph $G'_s = (V, E_0 \cup E'_s)$, where $E'_s \subset E_a$ is the set of original augmentation edges corresponding to $S\backslash\{e\}$, remains biconnected. Using the algorithm from Tarjan (1972), the biconnectivity-check can be performed in time $O(|V| + |S|)$. However, experimental results have shown that in the memetic algorithm, this alternative redundancy-check is particularly on larger problem instances significantly slower than the originally proposed one. The explanation lies in the fact that in locally optimal solutions, $|S|$ is typically substantially smaller than $|V_T|$, since several cut-points can often be covered by a single augmentation edge. Since we apply local improvement only to candidate solutions obtained from the initialization, recombination, or mutation procedures, and these solutions do not usually have many redundant edges, $|S| \ll |V_T|$ also holds in most of our cases. On the other hand, when considering local improvement without the memetic algorithm framework and the number of augmentation edges $|S|$ may be large, the redundancy-check using Tarjan's algorithm would presumably be more efficient.

4.3 Initialization

A solution of the initial population is created by starting with an empty edge-set S. Iteratively, an edge is randomly selected from E_A and included in S if it is not redundant. This process is repeated until all cut-nodes are completely covered, thus, the augmented graph G_s is biconnected.

Intuitively, cheaper edges appear in optimum solutions more likely than expensive edges. Therefore, the selection of edges for inclusion is biased toward cheaper edges according to a scheme originally proposed in Raidl (2000) for the selection of edges to be included by mutation in candidate solutions to the degree-constrained minimum spanning tree problem: During preprocessing, the edges in E_A are sorted according to costs. In this way, each edge has a rank, with ties broken randomly. A rank, thus an edge, is selected by sampling the random variable

$$rank = \lfloor |\mathcal{N}(0,s)|\,|V_T| \rfloor \bmod |E_A| + 1\,, \tag{3}$$

where $\mathcal{N}(0,s)$ is a normally distributed random variable with zero mean and standard deviation s, a strategy parameter controlling the strength of the scheme's bias toward cheap edges.

A solution created in this way is not necessarily locally optimal since the inclusion of an edge may make previously included edges redundant. Therefore, the memetic algorithm applies local improvement also to each initial solution.

4.4 Recombination

The recombination operator was designed with the aim to provide highest possible heritability, i.e. an offspring should consist of edges from its two parental solutions only. In the first step, edges common in both parents S_1 and S_2 are always adopted: $S \leftarrow S_1 \cap S_2$. Then, while not all cut-nodes are completely covered, an edge is selected from the set of remaining parental edges $(S_1 \cup S_2) \setminus S$ and included in the offspring S if it is not redundant. To emphasize the inclusion of low-cost edges again, they are selected via binary tournaments with replacement.

Figure 8 shows the recombination in pseudo-code. The check, whether an edge e actually helps in covering a cut-node—thus, if e is not redundant—can be performed efficiently in nearly constant amortized time when union-find data structures are maintained for all cut-nodes of degree greater than three. Compare the check whether a set of augmentation edges covers a cut-node described

in Sect. 3.1. The computational effort of the whole recombination procedure is $O((|S_1| + |S_2|)\,|V_T|)$ in the worst case and $O((|S_1| + |S_2|)\log|V_T|)$ on average.

4.5 Edge-Delete Mutation

The aim of mutation is to introduce new edges not appearing in the population into candidate solutions. Fig. 9 shows the mutation procedure in pseudo-code. From the candidate solution S, an edge e is selected and removed. That way, one or more cut-nodes from $\Psi(e)$ become uncovered. These uncovered cut-nodes are identified and processed in random order: For each such cut-node v_c, the edges from $\Gamma(v_c)$ are considered in random order and included in S if they help in reestablishing the cover of v_c, i.e., if they connect two yet unconnected cut-components of v_c.

The selection of the edge to be removed is biased toward more expensive edges by performing a binary tournament with replacement on S. The new edges to be included in S for reestablishing biconnectivity are chosen in an unbiased way to not reduce the population's diversity too much.

As initialization and recombination, this procedure does not guarantee to yield a locally optimal solution. Therefore, the memetic algorithm applies local improvement also after mutation. An upper bound for the worst-case computational effort of mutation is $O(|V_T|\,|E_A|)$. However, mutation is substantially faster in practice, and the time needed for local improvement dominates the time for mutation.

5 Empirical Results

To test the presented memetic algorithm and to compare it with previous approaches, problem instances of different size and structure were used. Since shrinking can always trivially reduce the problem of augmenting a general connected graph G_0 to the problem of augmenting a tree, we consider here only instances in which the fixed graph G_0 is a spanning tree. The used test instances were adopted from the following two sources.

- Random instances created by means of Zhu's generator[1]:

  Table 1 shows the characteristics of 27 instance-groups `A1` to `R2`, each consisting of 30 different instances. We call them *random instances*, since they were randomly created by a program

[1] Available at `www.ads.tuwien.ac.at/research/NetworkDesign/Augmentation`.

from Zhu (1999): Starting from $|V|$ nodes, edges are created between each pair of nodes $u, v \in V$, $u \neq v$, with the probabilities listed in column *dens*, the density of the graph. If the resulting graph is not biconnected, the creation is restarted. A random spanning tree is then determined on the graph yielding the set of fixed edges E_0. All other edges form set E_a and get assigned randomly chosen integer costs from the intervals listed in column $cost(e)$.

Note that instances with the same names `A1` to `R2` and the same characteristics have already been used in previous works (Zhu et al., 1999, Ljubić and Kratica, 2000, Raidl and Ljubić, 2002), however with only one representative instance per group instead of 30. Column $|E_a|$ of Table 1 lists the average numbers of augmentation edges and column CP(G_0) the average numbers of cut-points.

- Instances derived from Reinelt's TSP-library (TSPLIB)[2]:

  The larger instances listed in Table 2 are adopted from real-world traveling salesman problems. `pr226`, `lin318`, `pr439`, and `pcb442` are of Euclidean type, meaning that nodes represent points in the Euclidean plane, edges exist between any two nodes, and edge costs are the Euclidean distances of the corresponding points rounded up to the nearest integer value. The largest instance `pa561` is not of Euclidean type; it is a complete graph with edge costs directly given by a matrix.

  Since all these instances represent complete base graphs G, and incomplete graphs are of particular interest, too, additional sparse instances `pr226-sp`, `lin318-sp`, `pr439-sp`, `pcb442-sp`, and `pa561-sp` were derived from the original TSPLIB-graphs by considering for each node the edges to its $\lceil |V| \cdot 10\% \rceil$ nearest neighbors only, i.e. the 10%-*nearest-neighbor graphs*. In case of instance `pr226-sp`, the 10%-nearest-neighbor graph turned out to be not biconnected, and the 15%-nearest-neighbor graph was used instead.

  For the Euclidean instances we further calculated Delaunay triangulations yielding additional sparse instances `pr226-dt`, `lin318-dt`, `pr439-dt`, and `pcb442-dt`.

  In all these cases, minimum spanning trees were chosen as fixed graphs G_0.

In their last six columns, Tables 1 and 2 show the results of the memetic algorithm's preprocessing: the numbers of nodes $|V_T|$, augmentation edges $|E_A|$, and cut-nodes CP(T) of the block-cut-graphs, the CPU-times t_{pre} for preprocessing (in seconds) and the savings factors CP(G_0)/CP(T) and

[2]Available at `www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95`.

$|E_a|/|E_A|$. In case of the random instances, these values are average values over the 30 instances per group. All experiments described in this section were performed on a Pentium-III/800MHz PC.

Preprocessing results document that the fixing of augmentation edges, which enables a shrinking of the block-cut-graph and therefore a reduction of cut-nodes, is highly effective in sparser graphs like those of groups `A1` to `C3`. In these cases, the numbers of cut-nodes could often be reduced to less than one half. As a consequence, also the numbers of augmentation edges could be substantially reduced. 16% of the instances from groups `A1` to `B4` could even be completely solved by preprocessing since it was able to reduce each block-cut graph to a single block-node.

On denser problem instances, no edges could be fixed, thus, the numbers of cut-nodes in the block-cut graphs are identical to the numbers of cut-points in the original graphs. However, edge-elimination was in these cases highly effective. On average over all instances, the number of augmentation edges that need to be considered for further optimization could be reduced to about a quarter of the edges in E_a.

The following setup was used for the memetic algorithm as it proved to be robust for many different classes of instances in preliminary tests: Population size $|P| = 800$; group size for tournament selection $k = 5$; parameter for biasing initialization to include cheaper edges $s = 2.5$; crossover probability $p_{\text{cro}} = 1$; mutation probability $p_{\text{mut}} = 0.7$. Each run was terminated when no new best solution could be identified during the last $\Omega = 10\,000$ iterations. On all the instances we considered, this criterion allowed the MA to converge so that only minor improvements in the quality of final solutions can be expected when prolonging the runs. Thus, the main goal was to find high-quality solutions, and running times were considered only secondary.

We compare the memetic algorithm (MA) to the heuristics from Khuller and Thurimella (1993) (KT), Zhu et al. (1999) (ZKR), and the genetic algorithm from Ljubić and Kratica (2000) (LK). These previous heuristics were implemented and applied as described in these works. Thus, the new enhanced preprocessing of the MA was not used by them.

For each instance, KT has been run with each leaf-node of the block-cut tree becoming once the root of the arborescence, and the best solution obtained in this way is regarded as KT's final solution to the instance.

ZKR could only be applied to the smaller random instances of groups `A1` to `N2` because of its high computational effort. (The total CPU-time of a run was limited to 20 000 seconds.) For instances of groups `M1` to `N2`, only 10% of all leaves were subsequently tried as root of the block-cut tree, while for the other instances, all leaves were considered.

The setup of LK was the same as described in Ljubić and Kratica (2000) except for the termination criterion, which was changed to be similar to that of the MA in order to ensure convergence: A run was terminated when no new best solution could be identified during the last 100 000 evaluations. (Note, however, that the number of evaluations of the MA and LK may not directly be compared due to the different computational complexities of the algorithms.)

Table 3 shows average results of the four approaches on the random instances. Each heuristic was run once on each of the 30 instances of each group. For reference purposes, we were able to solve all these instances also to guaranteed optimality by a not yet published branch-and-cut-and-price approach. This exact algorithm relies on the MA, since it uses its high-quality solutions as starting solutions and initial bounds. The needed time and space resources were excessive and in particular much higher than those of the MA (except for the small instances). Due to its exponential computational effort, the approach has clear limits regarding the size and complexity of the instances to which it can be applied.

Column $|E_s^*|$ lists average numbers of edges in these optimal solutions. The qualities of the solutions E_s obtained by the algorithms are reported as percentage gaps with respect to the optimal costs $cost(E_s^*)$:

$$\%\text{-}gap = \frac{cost(E_s) - cost(E_s^*)}{cost(E_s^*)} \cdot 100\% \,. \tag{4}$$

Standard deviations of average gaps (σ) are also presented in the table. For LK and MA, average CPU-times and numbers of evaluated solutions until the best solutions were found (t, respectively *evals*), and success rates (sr), i.e. the percentage of instances for which optimal solutions could be found, are reported in addition. CPU-times include preprocessing: in case of KT, ZKR, and LK the derivation of the block-cut graph according to Sect. 1, in case of MA additionally the more sophisticated reductions and the creation of supporting data structures—in particular Γ and Ψ—according to Sect. 3.

Results show that MA clearly outperformed the other heuristics in most cases. It could find optimal solutions to all instances of groups `A1` to `D4` and `M1` to `M3`. On the remaining random instances, MA

was able to identify high-quality solutions with an average gap of only 0.33%. KT yielded in all cases the worst results. Among ZKR and LK, ZKR could usually identify slightly better solutions. Quality differences become most apparent in groups `M1` to `R2`.

Regarding the running times, KT was usually fastest (about 2 to 3 times faster than the times reported for MA), followed by MA. LK was usually much slower, in particular on the larger instances. ZKR needed in any case the most time. With over 15 000 seconds CPU-time for instances of group `N2`, ZKR is definitely only suitable for small instances.

Table 4 shows results for the larger TSPLIB-derived instances. Optimum solutions could be found by branch-and-cut only up to instance `pcb442-sp`. Total costs of these optimum solutions—or if unknown best-known solution values—and the numbers of edges in those solutions are listed in columns $cost(E_s^*)$ and $|E_s^*|$, respectively.

On these TSPLIB-derived instances, ZKR never terminated within the allowed maximum time of 20 000 seconds, and LK could obtain meaningful results on the eight sparse Euclidean instances only. Because of the stochastic nature of LK and MA, these heuristics were performed 30 times on each considered instance and Table 4 prints average results for them.

In contrast to ZKR and LK, MA scales well to the larger instances. Its CPU-time increases only moderately with the problem size due to the relatively low computational complexities of local improvement, recombination, and mutation. Because of the data structures created during preprocessing, MA required up to 420MB main memory for the largest instance `pa561` with $|E_a| = 156\,520$ augmentation edges. MA's solutions are of high quality again: On average, the gap was only 0.65%, and optimum or best-known solutions could be found several times. KT followed far behind with gaps between 19.6% and 32.6%; LK's results were even worse: its average gaps are all larger than 20.9%.

Statistical t-tests were performed and indicate that the quality differences between MA's solutions and those of the other approaches are significant at a 0.1% error-level on each instance. This also holds for the results on random instances shown in Table 3, except in those cases were also ZKR was able to identify always optimal solutions.

Figure 10 shows three exemplary solutions to the Euclidean problem instance `lin318-sp` found by KT, LK, and MA. Obviously redundant edges, as they are contained in the solution of KT, can never appear in a solution of MA due to its local improvement procedure.

The proposed preprocessing of MA can also be adapted to work with KT, ZKR, and LK. Tests we performed indicate that in particular the total running times of these approaches are reduced significantly in this way. However, the quality of obtained solutions was not substantially higher. On average over all random and TSPLIB-derived instances where the individual approaches terminated within the allowed time of 20 000 seconds, the total times were reduced by the factors 0.62 in case of KT, 0.81 in case of ZKR, and 0.27 in case of LK. The %-gaps were reduced on average by the factors 0.92 in case of KT, 0.96 in case of ZKR, and 0.91 in case of LK. On several instances of groups `A1` to `C4`, the combinations of our preprocessing with KT, ZKR, and LK were able to identify optimal solutions as the MA did. Nevertheless, on the larger and more complicated instances, these approaches were still not competitive with the MA.

To further investigate the difficulty of the problem instances and the effects of local improvement, we performed fitness-distance correlation analyses according to Jones and Forrest (1995) and Merz and Freisleben (1999, 2000). For each problem instance, 10 000 candidate solutions were created randomly and locally improved as in the initialization of the MA. These solutions were evaluated and their distances to the optimum solution in the search space were calculated. As distance metric, the size of the symmetric difference of the corresponding edge sets was used. Figure 11 shows fitness-distance plots for the first instance of group `R2` and instance `pr439`. The plots for the other instances have similar structure. Each point in these plots represents one locally optimal solution; the global optima are located at the lower left corners (point 0/0). In addition, Table 5 shows fitness-distance correlation coefficients ρ for ten of the largest instances with known global optima.

In all considered instances, the fitness is clearly correlated with the distance to the optimum ($0.51 \leq \rho \leq 0.71$), which is a general indication that an evolutionary algorithm might work efficiently on these instances. Furthermore, all local optima are plotted near to each other and have a relatively large distance to the optimum. This shows that simply creating random solutions and locally improving them by our method is not effective for its own. It does not imply that the local optima are also grouped together in the search space and the global optima are located far away from them. Table 5 also shows average distances of locally improved random solutions to the optima ($\overline{d_{\text{opt}}}$) and average distances between locally improved random solutions ($\overline{d_{\text{loc}}}$). Since $\overline{d_{\text{opt}}}$ is significantly smaller than $\overline{d_{\text{loc}}}$ for each instance, we can argue that the global optimum lies more or less in the center of the space of all locally optimal solutions. In Merz and Freisleben (1999), problems with

such a characteristic are said to have a *big valley* structure, and recombination operators preserving properties common to both parents can be expected to work well.

In the last two columns, Table 5 lists average probabilities PD_{cross} and PD_{mut} with which recombination, respectively mutation, followed by local improvement produces a candidate solution being identical to (one of) its parent solution(s). These probabilities were measured over complete runs of the MA. High values would indicate that the investigated variation operator does not work efficiently and it might be omitted without decreasing the overall effectiveness of the search significantly. In our case, these probabilities are always smaller than 18.5%. Thus, the variation operators in combination with local improvement successfully create new solutions in more than four out of five cases. In particular on dense base graphs such as the complete Euclidean problem instances, the probability of mutation leading to the same local optimum is very small: $PD_{\text{mut}} \leq 2.3$.

Table 6 further illustrates the importance of using both, recombination and mutation, and that it is not necessary to apply local improvement immediately after each variation operator. Shown are results for the following three variants of the MA: In MA-CLML, recombination and mutation are used, and local improvement is performed after each operator. In MA-CL, new candidate solutions are created only by recombination followed by local improvement. MA-ML applies always only mutation followed by local improvement. All strategy parameters were set identical as in the previous experiments with the only exception that in MA-ML, the probability of applying mutation was $p_{\text{mut}} = 1$. The performance values of these variants can therefore directly be compared to those of the original MA in Table 4.

MA-CL converged fastest, but the obtained solutions were in nearly all cases substantially poorer than those of the original MA. This points out the particular importance of mutation. MA-ML, on the other side, generally needed much more evaluations and also more time to converge. In particular on dense problem instances, MA-ML's solutions are far worse than those of the original MA.

Performance values of MA and MA-CLML are similar for nearly all instances. Only on the single instance `pa561`, MA yielded substantially better solutions than MA-CLML. No statistically significant differences can be observed for MA and MA-CLML in their numbers of needed evaluations and running times. We conclude that the question whether local improvement should be applied once per candidate solution or once after each variation operator is of minor importance.

6 Conclusions

The main features of the proposed memetic algorithm for the vertex-biconnectivity augmentation problem are: The effective deterministic preprocessing which reduces the search space in most cases substantially, the local improvement procedure which guarantees local optimality with respect to the number of augmentation edges of any candidate solution, and the strong heritability and locality of the proposed recombination, respectively mutation. Furthermore, the biasing of initialization and recombination to include low-cost edges more likely, respectively the biasing of mutation to remove more expensive edges more likely, play significant roles.

Supporting data structures established during preprocessing allow efficient implementations of initialization, recombination, mutation, and local improvement (Merz, 2000). Empirical tests indicate that the algorithm calculates solutions of high quality, which are optimal in many cases and usually significantly better than those of the other three heuristics from the literature. Although a theoretical upper bound for the computational costs of preprocessing is $O(|V|^2|E|)$, it is in practice also efficient on large problem instances and the memetic algorithm usually dominates the total computation time. Within the memetic algorithm, local improvement dominates the computational costs. The theoretical worst-case time complexity of locally improving one solution is $O(|V_T|^3)$, however, we have argued that the expected costs are substantially smaller. Empirical results support this and show that the approach scales well to instances of large size.

Notes

* This work is supported by the Austrian Science Fund (FWF) under the grant P13602–INF.

References

Aho, A. V., J. E. Hopcroft, and J. D. Ullman (1983). *Data Structures and Algorithms*. Reading, MA: Addison-Wesley.

Blickle, T. (1997). "Tournament Selection." In *Handbook of Evolutionary Computation*, T. Bäck, D. B. Fogel, and Z. Michalewicz, eds., New York: Oxford University Press. C2.3:1–C2.3:4.

Cheriyan, J., S. Vempala, and A. Vetta (2001). "An Approximation Algorithm for the Minimum-Cost k-Vertex Connected Subgraph." Submitted for journal publication.

Eswaran, K. P. and R. E. Tarjan (1976). "Augmentation Problems." *SIAM Journal on Computing* 5(4), 653–665.

Fialgo, S. and P. Mutzel (1998). "A New Approximation Algorithm for the Planar Augmentation Problem." In *Proceedings of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms*. ACM-SIAM.

Fonseca, C., J.-H. Kim, and A. Smith (eds.) (2000). *Proceedings of the 2000 IEEE Congress on Evolutionary Computation*. IEEE Press.

Fortz, B. (2000). *Design of Survivable Networks with Bounded Rings*. Network Theory and Applications. Universitè Libre de Bruxelles, Bruxelles, Belgium: Kluwer Academic Publishers.

Frederickson, G. N. and J. Jájá (1981). "Approximation Algorithms for Several Graph Augmentation Problems." *SIAM Journal on Computing* 10(2), 270–283.

Frederickson, G. N. and J. Jájá (1982). "On the Relationship between the Biconnectivity Augmentation and Traveling Salesman Problems." *Theoretical Computer Science* 19(2), 203–218.

Gabow, H. N., Z. Galil, T. Spencer, et al. (1986). "Efficient Algorithms for Finding Minimum Spanning Trees in Undirected and Directed Graphs." *Combinatorica* 6(2), 109–122.

Hsu, T.-S. and V. Ramachandran (1993). "Finding a Smallest Augmentation to Biconnect a Graph." *SIAM Journal on Computing* 22(5), 889–912.

Ishii, T. (2000). *Studies on Multigraph Connectivity Augmentation Problems*. Ph.D. thesis, Dept. of Applied Mathematics and Physics, Kyoto University, Kyoto, Japan.

Jones, J. and S. Forrest (1995). "Fitness Distance Correlation as a Measure of Problem Difficulty for Genetic Algorithms." In *Proceedings of the Sixth International Conference on Genetic Algorithms*, L. J. Eshelman, ed. Morgan Kaufmann, 184–192.

Kersting, S., G. R. Raidl, and I. Ljubić (2002). "A Memetic Algorithm for Vertex-Biconnectivity Augmentation." In *Applications of Evolutionary Computing: EvoWorkshops 2002*, S. Cagnoni, J. Gottlieb, E. Hart, et al., eds. Springer, volume 2279 of *LNCS*, 102–111.

Khuller, S. (1997). "Approximation Algorithms for Finding Highly Connected Subgraphs." In *Approximation Algorithms for NP-hard Problems*, D. S. Hochbaum, ed., Boston, MA: PWS. 236–265.

Khuller, S. and R. Thurimella (1993). "Approximation Algorithms for Graph Augmentation." *Journal of Algorithms* 14(2), 214–225.

Ljubić, I. and J. Kratica (2000). "A Genetic Algorithm for the Biconnectivity Augmentation Problem." In Fonseca et al. (2000), 89–96.

Ljubić, I. and G. R. Raidl (2001). "An Evolutionary Algorithm with Hill-Climbing for the Edge-Biconnectivity Augmentation Problem." In *Applications of Evolutionary Computing: EvoWorkshops 2001*, E. J. W. Boers, S. Cagnoni, J. Gottlieb, et al., eds. Springer, volume 2037 of *LNCS*, 20–29.

Merz, P. (2000). *Memetic Algorithms for Combinatorial Optimization Problems: Fitness Landscapes and Effective Search Strategies*. Ph.D. thesis, Department of Electrical Engineering and Computer Science, University of Siegen, Siegen, Germany.

Merz, P. and B. Freisleben (1999). "Fitness Landscapes and Memetic Algorithm Design." In *New Ideas in Optimisation*, Berkshire, England: McGraw-Hill. 245–260.

Merz, P. and B. Freisleben (2000). "Fitness Landscape Analysis and Memetic Algorithms for the Quadratic Assignment Problem." *IEEE Transactions on Evolutionary Computation* 4(4), 337–352.

Michalewicz, Z. (1996). *Genetic Algorithms + Data Structures = Evolution Programs*. Berlin: Springer.

Monma, C. L. and D. F. Shallcross (1989). "Methods for Designing Communications Networks with Certain Two-Connected Survivability Constraints." *Operations Research* 37(4), 531–541.

Moscato, P. (1999). "Memetic Algorithms: A Short Introduction." In *New Ideas in Optimization*, D. Corne, M. Dorigo, and F. Glover, eds., Berkshire, England: McGraw Hill. 219–234.

Raidl, G. R. (2000). "An Efficient Evolutionary Algorithm for the Degree-Constrained Minimum Spanning Tree Problem." In Fonseca et al. (2000), 104–111.

Raidl, G. R. and J. Gottlieb (1999). "On the Importance of Phenotypic Duplicate Elimination in Decoder-Based Evolutionary Algorithms." In *Late Breaking Papers at the 1999 Genetic and Evolutionary Computation Conference*, S. Brave and A. S. Wu, eds. Orlando, FL, 204–211.

Raidl, G. R. and I. Ljubić (2002). "Evolutionary Local Search for the Edge-Biconnectivity Augmentation Problem." *Information Processing Letters* 82(1), 39–45.

Stoer, M. (1992). *Design of Survivable Networks*, volume 1531 of *Lecture Notes in Mathematics*. Springer.

Tarjan, R. E. (1972). "Depth First Search and Linear Graph Algorithms." *SIAM Journal of Computing* 1, 146–160.

Watanabe, T. and A. Nakamura (1987). "Edge-Connectivity Augmentation Problems." *Journal of Computer and System Sciences* 35(1), 96–144.

Zhu, A. (1999). "A Uniform Framework for Approximating Weighted Connectivity Problems." B.Sc. thesis at the University of Maryland, MD.

Zhu, A., S. Khuller, and B. Raghavachari (1999). "A Uniform Framework for Approximating Weighted Connectivity Problems." In *Proceedings of the 10th ACM-SIAM Symposium on Discrete Algorithms*. 937–938.

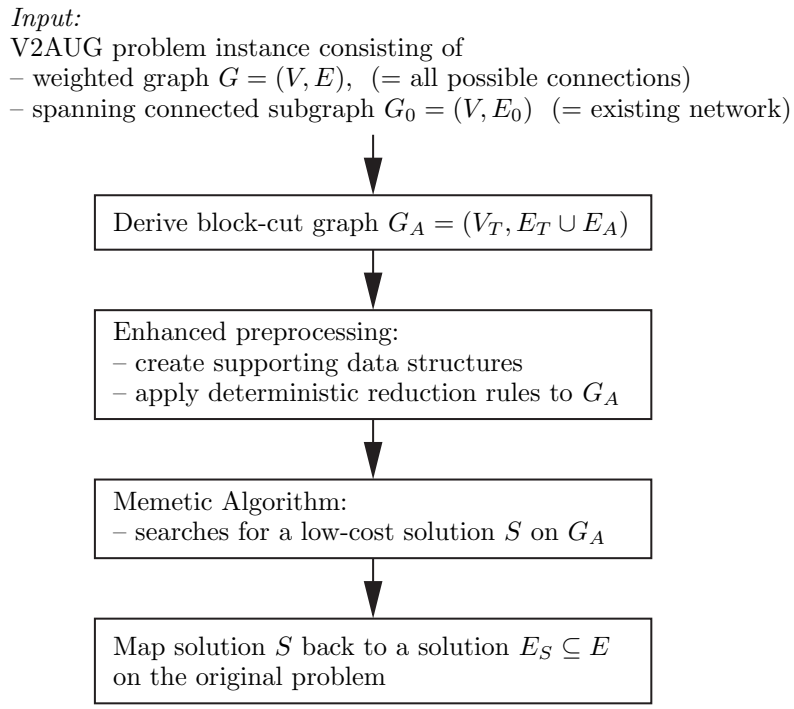
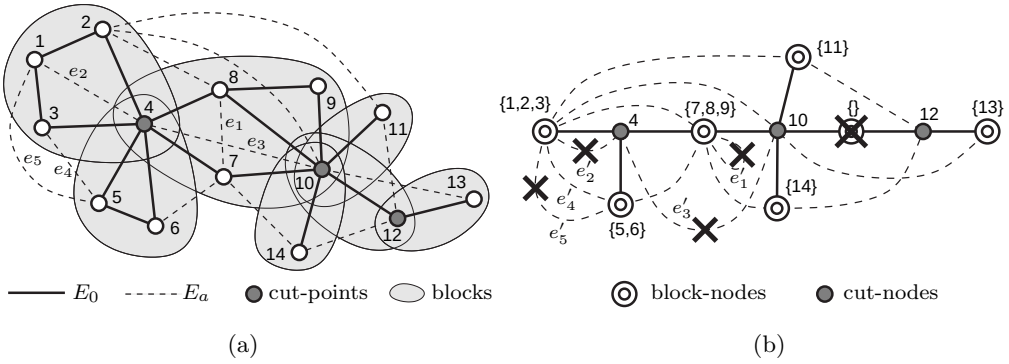


Figure 1: Basic structure of the proposed approach.



(a) (b)

Figure 2: (a) A base graph G with its fixed edges E_0 and optional augmentation edges E_a and (b) the corresponding block-cut tree T with the superimposed augmentation edges E_A.

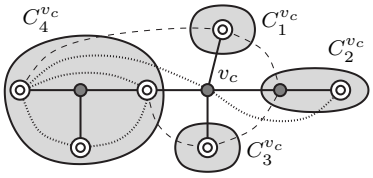


Figure 3: The cut-components $C_1^{v_c}$, $C_2^{v_c}$, $C_3^{v_c}$, and $C_4^{v_c}$ of a cut-node v_c. Dashed edges form edge-set $\Gamma(v_c)$. Dotted augmentation edges do not contribute in covering v_c.

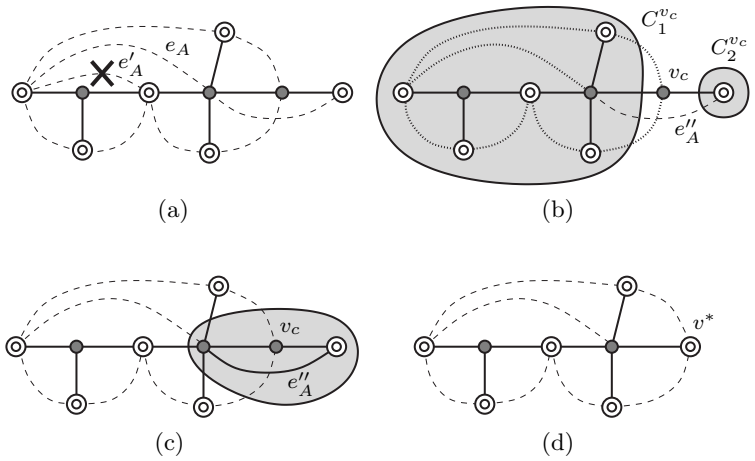


Figure 4: Reducing the block-cut graph: (a) Assuming $cost(e_A) \leq cost(e'_A)$, edge e'_A can be eliminated. (b) e''_A is the only edge able to connect the cut-components $C_1^{v_c}$ and $C_2^{v_c}$ of v_c and is therefore fixed. This introduces a new biconnected component in T (c), which is shrinked into the single new block-node v^* (d).

Memetic Algorithm for V2AUG:
begin
  create random initial population P of feasible solutions;
  for each solution $S \in P$ **do**
    locally improve (S);
  repeat
    select two parents $S_1, S_2 \in P$ via k-ary tournament selection;
    $S \leftarrow$ recombine (S_1, S_2);
    with probability p_{mut}: mutate (S);
    locally improve (S);
    if $S \notin P$ **then**
      replace the worst solution from P with S;
  until no new best solution found for Ω iterations;
end

Figure 5: The memetic algorithm for V2AUG.

procedure locally improve (S):

begin

$R \leftarrow S$; // set of edges to be considered for removal

// look for obviously essential edges:

for each $e_T \in E_T$ **do**

$\mathit{covered}[e_T] \leftarrow \emptyset$;

for each $e \in S$ **do**

for each $v_c \in \Psi(e)$ **do**

$\mathit{covered}[e_{T1}^{v_c}(e)] \leftarrow \mathit{covered}[e_{T1}^{v_c}(e)] \cup \{e\}$;

$\mathit{covered}[e_{T2}^{v_c}(e)] \leftarrow \mathit{covered}[e_{T2}^{v_c}(e)] \cup \{e\}$;

// remove obviously essential edges from R:

for each $e_T \in E_T$ **do**

if $|\mathit{covered}[e_T]| = 1$ **then**

$R \leftarrow R \setminus \mathit{covered}[e_T]$;

// check remaining edges in R:

for each $e \in R$ in decreasing *cost*-order **do**

$S \leftarrow S \setminus \{e\}$;

if $\exists v_c \in \Psi(e)$: v_c is uncovered in G_s **then**

$S \leftarrow S \cup \{e\}$;

end

Figure 6: The local improvement procedure.

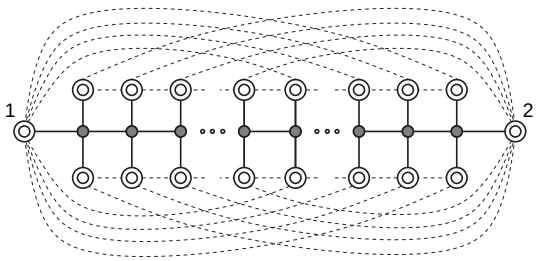


Figure 7: Worst case example for local improvement.

procedure recombine (S_1, S_2):

begin

$S \leftarrow S_1 \cap S_2$;

$R \leftarrow (S_1 \cup S_2) \setminus S$;

repeat

select $e \in R$ via binary tournament selection;

$R \leftarrow R \setminus \{e\}$;

if $\exists v_c \in \Psi(e)$: edge e helps in covering v_c **then**

// edge e is not redundant

$S \leftarrow S \cup \{e\}$;

until all cut-nodes $v_c \in V_T$ are covered;

return S;

end

Figure 8: Recombination.

procedure mutate (S):

begin

select $e \in S$ via binary tournament selection;

$S \leftarrow S \setminus \{e\}$;

for each uncovered $v_c \in \Psi(e)$ in random order **do**

$F \leftarrow \Gamma(v_c)$;

while v_c is uncovered **do**

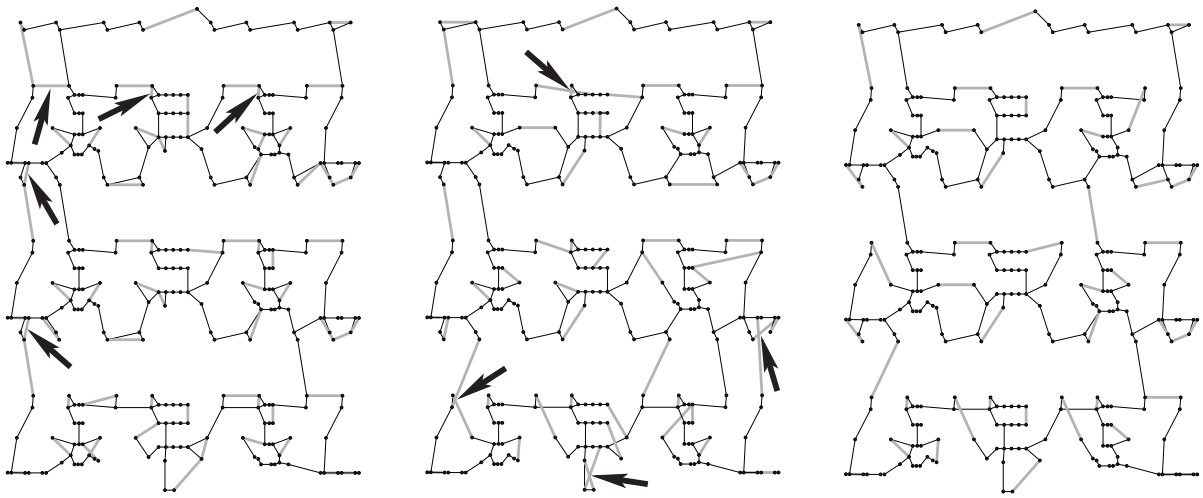
pick $e' \in F$ randomly; $F \leftarrow F \setminus \{e'\}$;

if e' helps in covering v_c **then**

$S \leftarrow S \cup \{e'\}$;

end

Figure 9: Edge-delete mutation.



(a) KT: %-$gap = 29.3$, $|E_s| = 69$ (b) LK: %-$gap = 35.8$, $|E_s| = 58$ (c) MA: %-$gap = 0.09$, $|E_s| = 45$

Figure 10: Exemplary solutions to problem instance `lin318-sp` found by (a) Khuller and Thurimella's heuristic, (b) the genetic algorithm from Ljubić and Kratica, and (c) the memetic algorithm. Solution edges are shown in gray. In (a), arrows mark obviously redundant edges. In (b), arrows mark obviously suboptimal crossing augmentation edges.

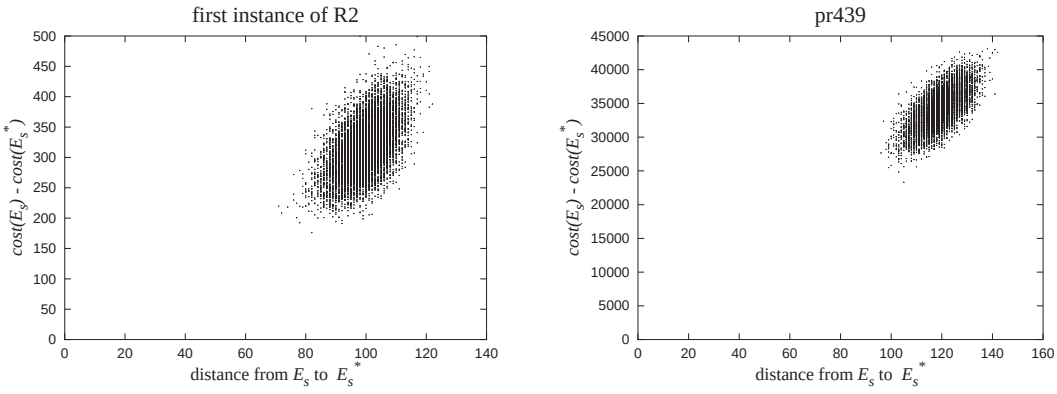


Figure 11: Fitness-distance plots for the first instance of group `R2` and instance `pr439`.

Table 1: Characteristics of instance-groups created with Zhu's generator and average results of the memetic algorithm's preprocessing.

Group	$\|V\|$	*dens*	$cost(e) \in$	$\|E_a\|$	$\mathrm{CP}(G_0)$	$\|V_T\|$	$\|E_A\|$	$\mathrm{CP}(T)$	t_{pre} [s]	$\mathrm{CP}(G_0)/\mathrm{CP}(T)$	$\|E_a\|/\|E_A\|$
A1	20	0.16	$\{1, 190\}$	18	11	8	5	4	<0.1	3.0	3.4
A2	30	0.10	$\{1, 435\}$	29	16	10	8	5	<0.1	3.2	3.8
A3	40	0.08	$\{1, 780\}$	37	22	9	6	4	<0.1	4.9	6.0
A4	30	0.12	$\{1, 435\}$	32	16	10	8	5	<0.1	3.4	4.2
A5	40	0.10	$\{1, 780\}$	46	23	20	18	10	<0.1	2.2	2.5
B1	60	0.05	$\{1, 1770\}$	54	35	14	10	7	<0.1	4.7	5.4
B2	20	0.50	$\{1, 190\}$	81	10	20	44	10	<0.1	1.0	1.8
B3	50	0.06	$\{1, 1225\}$	45	29	14	11	7	<0.1	4.1	4.1
B4	50	0.08	$\{1, 1225\}$	61	27	26	26	14	<0.1	2.0	2.3
B5	60	0.07	$\{1, 1770\}$	74	33	29	29	16	<0.1	2.1	2.6
B6	70	0.06	$\{1, 2415\}$	96	39	45	48	24	0.1	1.6	2.0
C1	80	0.06	$\{1, 3160\}$	113	43	50	55	26	0.1	1.6	2.1
C2	90	0.05	$\{1, 4005\}$	125	50	53	55	29	0.1	1.7	2.3
C3	100	0.05	$\{1, 4950\}$	153	54	63	73	34	0.1	1.6	2.1
C4	30	0.50	$\{1, 435\}$	191	15	30	107	15	<0.1	1.0	1.8
D1	70	0.15	$\{1, 2415\}$	279	37	70	196	37	<0.1	1.0	1.4
D2	40	0.50	$\{1, 780\}$	349	20	40	187	20	<0.1	1.0	1.9
D3	90	0.15	$\{1, 4005\}$	507	46	90	366	46	0.1	1.0	1.4
D4	80	0.15	$\{1, 3160\}$	396	41	80	284	41	0.1	1.0	1.4
D5	100	0.15	$\{1, 4950\}$	648	52	100	464	52	0.1	1.0	1.4
M1	70	0.15	$\{10, 1000\}$	284	37	70	207	36	<0.1	1.0	1.4
M2	80	0.15	$\{10, 1000\}$	388	42	80	278	42	<0.1	1.0	1.4
M3	90	0.15	$\{10, 1000\}$	492	46	90	352	46	0.1	1.0	1.4
N1	100	0.25	$\{11, 50\}$	1124	50	100	705	50	0.1	1.0	1.6
N2	110	0.25	$\{11, 50\}$	1384	56	110	874	56	0.1	1.0	1.6
R1	200	0.50	$\{1, 100\}$	9734	117	200	3888	117	4.3	1.0	2.5
R2	200	0.50	$\{5, 100\}$	9744	118	200	3852	118	3.5	1.0	2.5

Table 2: Instances derived from the TSPLIB and results of the memetic algorithm's preprocessing.

Instance	$\|V\|$	Type	$\|E_a\|$	$\mathrm{CP}(G_0)$	$\|V_T\|$	$\|E_A\|$	$\mathrm{CP}(T)$	t_{pre} [s]	$\mathrm{CP}(G_0)/\mathrm{CP}(T)$	$\|E_a\|/\|E_A\|$
pr226-dt	226	Euclidean	947	192	226	617	192	1.4	1.0	1.5
pr226-sp	226	Euclidean	3987	187	226	2550	187	1.7	1.0	1.6
pr226	226	Euclidean	25200	187	226	7089	187	11.0	1.0	3.6
lin318-dt	318	Euclidean	1563	248	318	1057	248	10.7	1.0	1.5
lin318-sp	318	Euclidean	5495	246	318	1874	246	12.6	1.0	2.9
lin318	318	Euclidean	50086	246	318	9473	246	95.6	1.0	5.3
pr439-dt	439	Euclidean	2156	358	439	1349	358	34.1	1.0	1.6
pr439-sp	439	Euclidean	11183	366	439	3026	366	42.7	1.0	3.7
pr439	439	Euclidean	95703	366	439	18700	366	280.2	1.0	5.1
pcb442-dt	442	Euclidean	2131	347	442	1298	347	42.9	1.0	1.6
pcb442-sp	442	Euclidean	10528	345	442	2557	345	53.7	1.0	4.1
pcb442	442	Euclidean	97020	345	442	19824	345	299.5	1.0	4.9
pa561-sp	561	matrix	18504	406	561	5175	406	157.2	1.0	3.6
pa561	561	matrix	156520	406	561	40601	406	417.6	1.0	3.9

Table 3: Results on random instances obtained by the heuristics from Khuller and Thurimella (KT) and Zhu et al. (ZKR), the genetic algorithm from Ljubić and Kratica (LK), and the memetic algorithm (MA).

Grp.	$\lvert E_s^* \rvert$	KT			ZKR			LK					MA				
		%-*gap*	σ	t [s]	%-*gap*	σ	t [s]	%-*gap*	σ	t [s]	*evals*	*sr* [%]	%-*gap*	σ	t [s]	*evals*	*sr* [%]
A1	6	1.2	2.2	0.1	0.0	0.0	0.8	0.5	1.6	<0.1	623	90.0	0.0	0.0	<0.1	586	100.0
A2	9	4.6	6.0	0.2	0.0	0.0	4.3	0.3	1.8	<0.1	2116	96.7	0.0	0.0	<0.1	666	100.0
A3	12	3.9	4.0	0.2	<0.1	0.1	14.2	0.0	0.0	0.1	4041	100.0	0.0	0.0	<0.1	506	100.0
A4	10	5.1	5.3	0.2	0.2	0.6	5.5	0.1	0.3	0.1	3038	93.3	0.0	0.0	<0.1	453	100.0
A5	12	7.7	6.4	0.2	<0.1	0.2	18.5	0.0	0.0	0.1	2640	100.0	0.0	0.0	<0.1	776	100.0
B1	18	4.4	4.2	0.4	0.1	0.5	73.5	<0.1	0.1	0.4	3788	96.7	0.0	0.0	<0.1	613	100.0
B2	7	4.5	6.7	0.1	0.0	0.0	6.1	<0.1	0.1	0.1	6103	96.7	0.0	0.0	0.1	887	100.0
B3	16	4.9	5.2	0.3	0.0	0.0	33.1	0.0	0.0	0.3	5951	100.0	0.0	0.0	<0.1	696	100.0
B4	16	6.2	5.6	0.3	0.1	0.4	64.4	0.2	0.8	0.4	5035	93.3	0.0	0.0	0.1	776	100.0
B5	19	4.9	4.2	0.4	0.1	0.3	131.9	0.2	0.9	0.7	8178	90.0	0.0	0.0	0.1	875	100.0
B6	22	7.5	4.4	0.7	<0.1	0.1	328.0	0.1	0.5	2.2	20615	86.7	0.0	0.0	0.3	1022	100.0
C1	26	6.8	4.5	0.9	<0.1	0.1	687.2	0.4	0.9	3.5	17681	80.0	0.0	0.0	0.4	1073	100.0
C2	29	7.6	4.2	1.1	0.2	0.6	1121.2	0.9	1.5	4.4	20166	60.0	0.0	0.0	0.5	1176	100.0
C3	32	8.9	5.6	1.6	0.1	0.2	2331.1	0.5	1.0	7.6	25078	53.3	0.0	0.0	0.6	1160	100.0
C4	11	4.6	7.3	0.3	<0.1	0.2	71.3	0.5	1.4	0.7	17303	80.0	0.0	0.0	0.4	1409	100.0
D1	23	7.6	4.5	1.1	0.2	0.5	1805.5	0.9	1.5	6.0	53353	43.3	0.0	0.0	1.7	1915	100.0
D2	14	5.0	5.0	0.6	0.1	0.2	403.0	0.4	1.1	2.4	31420	86.7	0.0	0.0	0.7	1838	100.0
D3	31	6.9	4.0	2.4	0.2	0.4	11046.5	1.1	1.2	15.9	51278	20.0	0.0	0.0	3.9	3016	100.0
D4	27	9.4	5.2	1.8	0.1	0.2	5208.8	1.6	2.0	11.2	49910	36.7	0.0	0.0	2.9	2686	100.0
D5	34	9.7	4.8	3.5	0.1	3.2	21762.5	1.8	2.2	25.5	59190	26.7	<0.1	0.1	5.9	4167	96.7
M1	23	8.5	4.9	1.2	0.4	0.9	237.0	1.7	2.1	6.7	49498	30.0	0.0	0.0	1.7	1984	100.0
M2	27	8.3	3.6	1.7	0.6	0.9	503.7	2.2	1.7	21.9	82105	10.0	0.0	0.0	2.9	2719	100.0
M3	30	9.8	4.5	2.4	0.3	0.6	1178.4	1.4	1.4	33.4	87141	26.7	0.0	0.0	4.3	3354	100.0
N1	27	34.4	6.5	5.2	3.2	1.7	14993.3	13.2	4.3	107.0	147146	0.0	0.2	0.3	9.5	8387	60.0
N2	29	36.5	5.7	7.0	4.1	2.3	16006.1	16.4	4.3	147.5	147030	0.0	0.4	0.5	13.7	11694	43.3
R1	57	16.3	3.8	64.3	–	–	–	10.7	4.4	4896.1	228723	0.0	0.1	0.2	39.8	9766	63.3
R2	47	31.9	4.2	91.6	–	–	–	20.2	4.7	5232.4	309073	0.0	0.4	0.4	58.5	21877	13.3

Table 4: Results on the TSPLIB-derived instances obtained by the heuristics from Khuller and Thurimella (KT) and Zhu et al. (ZKR), the genetic algorithm from Ljubić and Kratica (LK), and the memetic algorithm (MA). Values marked by '*' are from best-known solutions, since optimum values are unknown.

Instance	$cost(E_s^*)$	$\lvert E_s^* \rvert$	KT		LK			MA				
			%-*gap*	t [s]	%-*gap*	σ	t [s]	%-*gap*	σ	t [s]	*evals*	*sr* [%]
pr226-dt	25152	25	19.6	1.4	26.6	8.8	47.8	0.0	0.0	3.9	1910	100.0
pr226-sp	22824	24	22.5	11.7	27.3	4.8	640.2	0.1	0.6	17.6	9800	96.7
pr226	22824	24	22.0	138.9	–	–	–	2.6	1.2	33.2	13073	16.7
lin318-dt	12013	45	28.1	5.0	20.9	2.3	246.7	<0.1	<0.1	26.0	9895	0.0
lin318-sp	11797	46	29.3	33.2	41.1	3.8	2633.7	0.3	0.3	40.8	27130	6.7
lin318	11797	46	29.3	620.4	–	–	–	1.0	0.5	128.9	23391	0.0
pr439-dt	28310	52	20.6	8.3	25.1	6.0	491.6	0.0	0.0	52.6	7557	100.0
pr439-sp	26800	48	21.2	71.0	40.5	4.3	13471.2	1.1	0.7	79.5	12164	20.0
pr439	26800	48	21.2	1498.5	–	–	–	2.5	1.1	408.1	22301	0.0
pcb442-dt	10328	60	31.4	12.3	21.4	3.1	320.5	0.1	0.1	67.6	9306	43.3
pcb442-sp	10460	60	32.6	106.5	33.8	5.2	18429.2	0.3	0.2	91.9	16902	6.7
pcb442	10478*	61*	30.5	2030.8	–	–	–	0.3	0.2	366.3	21493	3.3
pa561-sp	782*	101*	31.1	303.0	–	–	–	0.3	0.1	250.0	20868	3.3
pa561	784*	101*	30.4	5482.7	–	–	–	0.4	0.4	599.8	34710	13.3

Table 5: Fitness distance correlation coefficient ρ, average distance $\overline{d_{\text{opt}}}$ of locally improved random solutions to the optimum, average distance $\overline{d_{\text{loc}}}$ between locally improved random solutions, and average probabilities PD_{cross} and PD_{mut} that recombination, respectively mutation, followed by local improvement produces a candidate solution being identical to a parent.

Instance	ρ	$\overline{d_{\text{opt}}}$	$\overline{d_{\text{loc}}}$	PD_{cross}	PD_{mut}
`N1` (1. instance)	0.57	53.4	62.7	15.3	14.1
`N2` (1. instance)	0.54	58.2	69.4	13.8	14.3
`R1` (1. instance)	0.51	107.5	116.9	13.0	3.4
`R2` (1. instance)	0.55	99.9	118.2	12.5	4.5
`pr226-sp`	0.53	66.6	71.5	17.0	12.6
`pr226`	0.52	66.8	71.0	18.4	2.0
`lin318-sp`	0.65	97.7	115.2	15.5	14.5
`lin318`	0.63	95.8	115.8	15.4	2.3
`pr439-sp`	0.71	113.9	129.5	14.0	10.2
`pr439`	0.68	119.4	125.8	13.4	1.3

Table 6: Performance of different MA-variants: applying local improvement after recombination and after mutation (MA-CLML), using only recombination followed by local improvement (MA-CL), and using only mutation followed by local improvement (MA-ML).

Grp./Inst.	MA-CLML					MA-CL					MA-ML				
	%-*gap*	σ	t [s]	*evals*	*sr*	%-*gap*	σ	t [s]	*evals*	*sr*	%-*gap*	σ	t [s]	*evals*	*sr*
N1	0.3	0.3	8.1	8391	46.7	0.6	0.6	7.2	6318	33.3	0.9	0.9	21.9	31825	20
N2	0.5	0.5	11.5	12023	36.7	1.3	1.3	7.1	5561	13.3	1.2	1.2	30	40819	6.7
R1	0.1	0.1	106.4	22558	43.3	0.4	0.4	22.7	4120	30	16	16.3	233.6	106740	0.0
R2	0.7	0.8	76	27587	3.3	1.3	1.4	22.8	5677	0.0	8.8	8.9	148.3	79119	0.0
pr226-dt	0.0	0.0	3.8	1843	100.0	<0.1	<0.1	3.5	1694	93.3	0.0	0.0	6.6	11073	100.0
pr226-sp	0.0	0.0	12.5	11777	100.0	22.9	4.4	13.8	5278	0.0	0.2	0.3	29.0	52534	46.7
pr226	2.4	1.4	30.7	24320	16.7	24.0	3.9	27.6	5367	0.0	8.2	2.7	43.8	56114	0.0
lin318-dt	0.2	0.5	27.7	10273	0.0	1.5	0.7	17.3	4013	0.0	0.1	0.3	34.7	34923	6.7
lin318-sp	0.3	0.3	41.0	35853	0.0	3.3	1.0	21.3	6771	0.0	1.6	1.1	54.0	73078	0.0
lin318	1.5	0.7	129.9	33955	0.0	3.5	1.6	111.0	6179	0.0	15.4	3.9	156.5	92059	0.0
pr439-dt	0.0	0.0	50.5	6626	100.0	2.0	1.8	45.1	3971	0.0	0.3	0.9	71.4	35827	50.0
pr439-sp	1.0	0.7	72.3	18073	23.3	5.4	1.8	68.7	5507	0.0	2.3	1.1	130.7	84829	0.0
pr439	2.0	0.8	376.1	44083	0.0	40.9	3.1	383.0	6373	0.0	19.9	5.7	457.0	98467	0.0
pcb442-dt	0.1	0.1	68.1	8981	56.7	1.5	0.9	54.4	3844	0.0	0.1	0.2	82.4	28030	56.7
pcb442-sp	0.3	0.2	95.2	24473	6.7	1.7	0.8	68.1	4805	0.0	1.6	0.9	187.9	106494	0.0
pcb442	0.7	0.4	365.3	28673	0.0	1.4	0.6	334.8	5120	0.0	33.5	7.5	464.0	106346	0.0
pa561-sp	0.4	0.2	275.4	30987	3.3	3.7	0.5	195.1	5802	0.0	5.9	1.3	462.2	129287	0.0
pa561	2.2	0.5	584.7	31361	0.0	3.8	0.6	493.9	5369	0.0	34.5	5.9	764.8	106649	0.0