A Memetic Algorithm for Vertex-Biconnectivity Augmentation

Sandor Kersting, Günther R. Raidl, and Ivana Ljubić

Institute of Computer Graphics and Algorithms, Vienna University of Technology,
Favoritenstraße 9–11/186, 1040 Vienna, Austria
{kersting|raidl|ljubic}@ads.tuwien.ac.at

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 an effective preprocessing of problem data and a fast local improvement strategy which is applied during initialization, mutation, and recombination. Only feasible, locally optimal solutions are created as candidates. Empirical results indicate the superiority of the new approach over two previous heuristic and an earlier evolutionary method.

1 Introduction

Robustness against failure is an important issue when designing a commercial computer network. It is often not acceptable that the failure of a single service node – be it a computer, router, or other device – leads to a disconnection of others. Redundant connections need to be established to provide alternative routes in case of a (temporary) break of any one node. We represent such a network by an undirected graph. It is said to be *vertex-biconnected* if at least two nodes need to be removed together with their incident edges in order to separate the graph into disconnected components. In a connected graph that is not vertex-biconnected a critical node whose removal would disconnect the graph is called *cut-point*. We say that we *cover a cut-point* when we add a set of edges to the graph which ensures that the removal of this vertex no longer disconnects the graph. Our global aim is to identify a set of edges with minimum total costs that covers all existing cut-points.

A more formal definition of the vertex-biconnectivity augmentation problem for graphs (V2AUG): 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 minimal.

The next section summarizes former approaches to this problem. Then, a new memetic algorithm is presented. Section 3 explains its preprocessing, which creates needed data structures and reduces the size of the problem in general considerably by fixing or eliminating certain edges in safe ways. Section 4 describes the main algorithm, including a new local improvement algorithm for creating locally optimal offsprings only. In Sect. 5 empirical results are presented and compared to two previous heuristics and a hybrid genetic algorithm. Conclusions are drawn in Sect. 6.

2 Previous Work

Eswaran and Tarjan [1] originally investigated the V2AUG problem. They 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 [4].

Frederickson and Jàjà [2] provided an approximation algorithm for the general case which finds a solution within a factor 2 of the optimum. The algorithm includes a preprocessing that transforms the fixed graph G_0 into a *block-cut tree*, see the Sect. 3. Each potential augmentation edge from E_a is superimposed on the block-cut tree, and certain redundant edges are identified and eliminated. In the main part of the algorithm, the block-cut tree is directed toward an arbitrarily chosen leaf as root node, and each tree-edge is assigned zero costs. Each cut-point is substituted by star-shaped structures 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. All superimposed augmentation edges are also directed. A minimum out-branching algorithm as described by Gabow et al. [3] is then applied to the block-cut tree including the superimposed augmentation edges to identify the solution's ede-set E_s. The computational effort of the algorithm is $O(|V|^2)$.

An improved variant of this approximation algorithm has been developed by Khuller and Thurimella [5]. It exhibits a time complexity of only $O(|E| + |V| \log |V|)$, but has the same approximation factor of 2.

An iterative approach based on Khuller and Thurimella's algorithm has been proposed by Zhu et al. [10]. 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 branching 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 branching 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 significantly better than when applying Khuller and Thurimella's algorithm. However, time requirements are raised significantly.

A straight-forward hybrid genetic algorithm for the V2AUG problem has been proposed by Ljubić and Kratica [6]. 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 by a greedy algorithm which temporarily removes cut-points one by one and searches for suitable augmentation edges that reconnect separated components.

Another, "lighter" kind of connectivity property is edge-biconnectivity, which means that a graph remains connected after the removal of any single edge. While vertex-biconnectivity implies edge-connectivity, this is not true vice versa. 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 special block-cut tree.

Recently, Raidl and Ljubić described in [7, 9] an effective evolutionary algorithm for E2AUG. A compact edge set encoding and special initialization and variation operators that include a local improvement heuristic are applied. In this way, the space of locally optimal solutions is searched only. The approach belongs to the broader class of so-called *local-search-based memetic algorithms* [8].

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

3 Preprocessing

During preprocessing, a block-cut tree is derived from the fixed graph G_0 according to [1], and other supporting data structures are created. They are all needed for an efficient implementation of the main algorithm. Furthermore, several deterministic rules are applied in order to reduce E_a in a safe way. The following paragraphs describe these mechanisms in detail.

3.1 The Block-Cut Tree

A block-cut tree $T = (V_T, E_T)$ with node set V_T and edge set E_T is an undirected tree that represents the connections between already vertex-biconnected components (called *blocks*) and cut-points of the underlying fixed graph G_0.

Fig. 1. The derivation of a block-cut tree: (a) given graph G_0, (b) identified blocks (shaded areas) and cut-points (square nodes), and (c) the block-cut tree.

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 block-node is associated with all nodes of the represented block in G_0 that are no cut-points. If the represented block consists of cut-points only, the block-node is not associated with any node from V_0.

A cut-node and a block-node are connected by an edge in E_T iff the corresponding cut-point is part of the block in G_0. Thus, cut-nodes and block-nodes always alternate on any path in T. The resulting structure is always a tree, since otherwise, the nodes forming a cycle can be shrinked into a single, larger block. Figure 1 illustrates the derivation of the block-cut tree.

After T has been derived from G_0, all potential 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. The mapping from E_A to E_a is stored in order to be finally able to derive the original edges of an identified solution. Note that $G_A = (V_T, E_T \cup E_A)$ may be a multi-graph containing self-loops and multiple edges between two nodes; however, the reductions described in Sect. 3.3 will make this graph simple.

3.2 When is a Cut-Point Covered?

A block-cut tree's edge $e \in E_T$ is said to be covered by an augmentation edge $e_A = (u, v) \in E_A$ iff 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 edges need to be covered, but this is in general not a sufficient condition.

If v_c would be removed from T, the tree will fall into k disconnected components $C_1^{v_c}, \ldots, C_k^{v_c}$, where k is the degree of v_c; we call them cut-components of v_c. We say an augmentation edge $e_A = (u, v) \in E_A$ *contributes in covering the cut-node* v_c, iff *two* edges incident to v_c are covered by e_A. Such an augmentation edge is obviously not incident to v_c and unites two cut-components $C_i^{v_c}$ and $C_j^{v_c}$. To completely cover v_c, exactly $k - 1$ augmentation edges are needed, and they must unite all components $C_1^{v_c}, \ldots, C_k^{v_c}$ into one.

Fig. 2. Examples for preprocessing: (a) Edges that do not contribute in covering a cut-point are removed. (b) When $cost(e6) \leq cost(e4)$ and $cost(e6) \leq cost(e5)$, $e4$ and $e5$ are discarded. (c) As $e6$ is the only edge that connects C_2^{v1} to any other cut-component of $v1$, it is fixed; (d) the cycle caused by fixing $e6$ is shrinked into a new block; (e) $e10$ becomes a self-loop and is finally also discarded.

For any cut-node v_c, let $A(v_c) \subseteq E_A$ be the set of all augmentation edges that contribute in covering v_c by uniting two of its cut-components. Furthermore, for each $e_A \in E_A$, let $R(e_A)$ be the set of all cut-nodes to whose covering e_A contributes. The proposed memetic algorithm explicitly computes and stores all sets $A(v_c)$ for all cut-nodes and the sets $R(e_A)$ for all augmentation edges as supporting data structures for its main part.

Later, we need to efficiently check if a certain cut-node is covered by a subset of augmentation edges $S \subset E_A$. This check is in general performed in $O(|S|)$ time with the aid of a union-find data structure. However, in most cases the degree of the cut-node is less than four, and then it is sufficient to just check whether each cut-component of the considered cut-node is connected to any other cut-component.

3.3 Reducing the Search Space

From E_A we discard all edges that do not contribute in covering any cut-node ($R(e_A) = \{\}$). In particular, edges forming self-loops or edges connecting a cut-node with an adjacent block-node or with another cut-node adjacent to the same block-node are removed in this way; see Fig. 2(a). Furthermore, from multiple edges connecting the same nodes in T, only one with minimum weight is retained. In this way, $G_A = (V_T, E_T \cup E_A)$ becomes a simple graph.

In addition to these simple reductions, we apply the following more sophisticated steps:

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 edges which are covered by e'_A (in addition to others), e'_A is obsolete and can be discarded. All such edges can be identified in $O(|V|^2)$ time

as a byproduct from a dynamic programming algorithm that computes distance values needed for the algorithm from Frederickson and Jájá [2]; see Fig. 2(b).

Fixing of Edges: An edge $e_A \in E_A$ must be included in any feasible solution to the V2AUG problem, when it represents the only possibility to connect a cut-node's cut-component $C_i^{v_c}$ to any other cut-component of v_c. In more detail, we process for each cut-node v_c its set $A(v_c)$ and look for such edges, which are then fixed by moving them from E_A to E_T; see Fig. 2(c). The corresponding original augmentation edges from E_a are permanently marked to be included in any future solution.

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 block-node, see Fig. 2(d) and 2(e). After shrinking all cycles, all changes in T are reflected to the supporting data structures. Due to these changes, more edges may become available for elimination. Therefore, all reduction steps are repeated until no further shrinking is possible.

4 The Memetic Algorithm

The main part of the new approach is a steady-state evolutionary algorithm, in which in each iteration one new candidate solution is always created by selecting two parents in k-ary tournaments with replacement, recombining them and applying mutation. Such a solution replaces the worst solution in the population with one exception: To maintain a minimum diversity, a new candidate that resembles a solution already contained in the population is discarded.

A solution is represented by directly storing the set of its augmentation edges $S \subseteq E_A$ in form of a hash-table. In this way, only $O(|S|) = O(|V|)$ space is needed, since $|S| < |V|$ in any locally optimal solution, and an edge can be added, deleted or checked for existence in constant time.

Local Improvement: For the creation of initial solutions and the variation operators that derive new solutions, the following local improvement method plays a central role. From a feasible solution S, it removes redundant edges until the solution becomes locally optimal in the sense that no further edge can be removed without including others or making the solution infeasible.

For the cut-components of each cut-node, it is first determined how often each of them is connected to any other by the edges in S. Edges that provide the only connection for any cut-component must always be included in a feasible solution and are therefore not redundant.

The remaining edges in S are then processed one-by-one in decreasing cost-order. All cut-nodes in whose coverage a certain augmentation edge $e \in S$ participates ($R(e)$) are checked if they remain covered when e is removed, see Sect. 3.2. If this is the case, this edge is actually redundant and removed from S.

In the worst case, the computational effort of this local improvement may be $O(|V|^2 \log |V|)$, however, it is much lower on average.

Initialization: A member of the initial population is created by randomly selecting edges from E_A without replacement and including each in the initially empty edge-set S if it is not redundant. This process stops when all cut-nodes are covered.

The selection of edges for inclusion is biased toward cheaper edges by sorting E_A according to costs, and choosing an edge via the following random-rank:

$$rank = \lfloor \mathcal{N}(0, s) \cdot |V| \rfloor \bmod |E_a| \,, \tag{2}$$

$\mathcal{N}(0, s)$ is a normally distributed random variable with zero mean and standard deviation s, a strategy parameter that determines the strength of biasing.

Recombination: This operator was designed with the aim to provide highest possible heritability. First, edges common in both parents S_1 and S_2 are always adopted: $S = S_1 \cap S_2$. Then, while not all cut-nodes are covered, an edges is randomly selected from the set of remaining parental edges $((S_1 \cup S_2) \setminus (S_2 \cap S_1))$ and included in the offspring S if it provides a new cover. To emphasize the inclusion of low-cost edges, they are selected via binary tournaments. As final step, local improvement is called.

Mutation: Having created a new offspring via recombination, mutation is applied with a certain probability in order to introduce new edges that were not available in the parents. From S, one edge is selected randomly and removed. This makes one or more cut-nodes uncovered. These cut-nodes are identified and newly covered in random order: For each cut-node v_c, the edges from $A(v_c)$ that would actually help in covering v_c anew are determined. From this set, edges are repeatedly chosen at random and included in S until v_c is completely covered. Finally, local improvement is applied again.

The selection of the edge to be removed is biased toward more expensive edges: A pair of edges is drawn at random and a cost-proportional selection among them decides which edge is actually removed. This technique is preferred here over traditional binary tournaments due to its less pressure.

5 Empirical Results

To compare the presented approach with other algorithms we have used test instances of different size and structure. Since shrinking can always reduce the problem of augmenting a general connected graph G_0 to the problem of augmenting a tree, G_0 is always a spanning tree in these test instances. Table 1 shows the number of nodes, the number of augmentation edges, and the number of cut-points (CP) before and after applying the memetic algorithm's preprocessing.

The first ten instances have been created with Zhu's generator [10] and were already used in [6]. The remaining ones are derived from Euclidean instances of Reinelt's TSP-library[1] in the following way: G is the graph containing all nodes

[1] www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95

Table 1. Considered problem instances and results of the memetic algorithm's preprocessing.

instance	$\|V\|$	$\|E_a\|$	$cost(e) \in$	CP(G_0)	$\|V_T\|$	$\|E_A\|$	CP(T)
B1	60	55	$\{1, 2, ..., 1770\}$	34	8	6	4
C3	100	149	$\{1, 2, ..., 4950\}$	53	59	67	33
M1	70	290	$\{0, 10, ..., 1000\}$	37	70	227	37
M2	80	327	$\{0, 10, ..., 1000\}$	41	80	242	41
M3	90	349	$\{0, 10, ..., 1000\}$	42	90	262	42
N1	100	1104	$\{10, 11, ..., 50\}$	50	100	687	50
N2	110	1161	$\{10, 11, ..., 50\}$	56	110	734	56
R1	200	9715	$\{1, 2, ..., 100\}$	115	200	3995	115
E1	200	19701	random Eucl.	154	200	4104	154
E2	300	11015	random Eucl.	192	300	4462	192
a280 (50)	280	7654	TSP-lib Eucl.	218	280	1561	218
a280 (100)	280	15769	TSP-lib Eucl.	220	280	3322	220
a280 (∞)	280	38781	TSP-lib Eucl.	217	280	10182	217
pr439 (100)	439	26865	TSP-lib Eucl.	364	439	6095	364
pr439 (200)	439	55111	TSP-lib Eucl.	362	439	11890	362
pr439 (∞)	439	95703	TSP-lib Eucl.	362	439	19574	362
rat575 (∞)	575	164451	TSP-lib Eucl.	436	575	34514	436
pcb1173 (30)	1173	18321	TSP-lib Eucl.	953	1173	4492	953
d2103 (50)	2103	55630	TSP-lib Eucl.	1963	2103	6657	1963

Table 2. Results of the heuristics from Khuller and Thurimella (KT), Zhu et al. (ZKR), the genetic algorithm from Ljubić and Kratica (LK), and the memetic algorithm (MA).

		KT	ZKR	LK		MA				
instance	C^*	*gap*	*gap*	*gap*	*evals*	*gap*	σ	t [s]	*evals*	SR [%]
B1	15512.0*	8.6	0.0	0.0	900	0.0	0.0	<1	800	100.0
C3	59129.0*	10.0	1.1	0.0	7300	0.0	0.0	<1	820	100.0
M1	2940.0*	8.2	0.0	0.0	2700	0.0	0.0	8	2144	100.0
M2	4600.0*	5.9	0.0	0.0	8700	0.0	0.0	10	1157	100.0
M3	4980.0*	6.2	0.4	0.0	8100	0.0	0.0	14	1508	100.0
N1	390.0*	31.5	4.1	4.9	27800	0.0	0.0	37	6264	100.0
N2	429.0*	39.2	5.8	2.3	90000	0.0	0.0	68	10980	100.0
R1	121.4*	16.8	–	–	–	0.5	0.6	818	38547	6.7
E1	2873.8*	21.1	–	–	–	0.7	0.3	88	44667	3.3
E2	9588.5	33.0	–	–	–	0.9	0.5	863	41385	3.3
a280 (50)	474.0*	25.1	–	–	–	0.1	0.1	28	5420	56.7
a280 (100)	473.0*	29.4	–	–	–	0.1	0.3	62	15487	76.7
a280 (∞)	490.0	21.6	–	–	–	1.3	0.8	103	24636	6.7
pr439 (100)	27907.0	20.5	–	–	–	0.5	0.4	246	34450	10.0
pr439 (200)	28518.0	18.1	–	–	–	0.9	0.5	353	35548	3.3
pr439 (∞)	27940.0	19.9	–	–	–	1.5	1.1	664	41361	3.3
rat575 (∞)	1558.0	32.3	–	–	–	1.9	1.1	1926	40331	3.3
pcb1173 (30)	11464.0	28.1	–	–	–	0.3	0.1	5552	77509	3.3
d2103 (50)	7333.0	9.6	–	–	–	0.0	0.0	15389	18885	10.0

of the TSP-instance and edges for each node to its nearest k neighbors, where k is the number shown in parentheses in Table 1; $k = \infty$ represents the complete graph. Edge costs are always the Euclidean distances rounded to nearest integer values. From G, a minimum spanning tree is derived and fixed as G_0.

Results of preprocessing document that a fixing of edges is only possible in shallower graphs like B1 and C3, where the number of cut-points could be dramatically reduced. In case of dense graphs, edge-elimination was highly effective. On average, the number of augmentation edges could be reduced to about a quarter.

The following setup was used for the memetic algorithm. Population size: 800; group size for tournament selection: 3; parameter s for biasing initialization toward cheaper edges: 0.5; mutation probability 0.7. Each run was terminated when no new best solution could be identified during the last $10,000$ iterations.

We compare the memetic algorithm, called MA, to the heuristics from Khuller and Thurimella [5] (KT), Zhu et al. [10] (ZKR), and the hybrid genetic algorithm from Ljubić and Kratica [6] (LK). For smaller instances, we were able to derive optimum solution values by a not yet published branch-and-cut approach. Table 2 shows in column C^* these optimum values marked by '*' or otherwise best-known solution values. For the heuristic approaches, the qualities of final solutions are reported as percentage gaps with respect to C^*: $gap = (cost(S) - C^*)/C^* \cdot 100\%$.

KT was run once for each leaf-node becoming the root of branching, and the best obtained gaps are shown. ZKR could only by applied to smaller instances due to its high computational effort. The same is true for LK, for which the shown gaps are adopted from [6]; they represent best values obtained from 10 runs per instance. MA's gaps are averaged over 30 runs for all instances, and σ shows the gaps' standard deviations. t gives the CPU-times on a PentiumIII/800MHz PC and *evals* the number of evaluated solutions until the finally best solutions had been identified. The success rate SR, finally, is the percentage of MA's runs that yielded optimum or best-known solutions.

It can be seen that KT performed generally worst. For the smaller instances, where results of ZKR and LK are available, MA found optimum solutions in any run. Furthermore, MA scaled well to larger instances. In all our test cases, it could identify solutions with gaps less than 2% with high reliability, as in particular also the small standard deviations document. The running times and needed numbers of evaluations increase only moderately with the problem size. Figure 3 shows two exemplary solutions.

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 many cases dramatically, the local improvement procedure which guarantees local optimality of any created candidate solution, and the the strong heritability and locality of the proposed recombination, re-

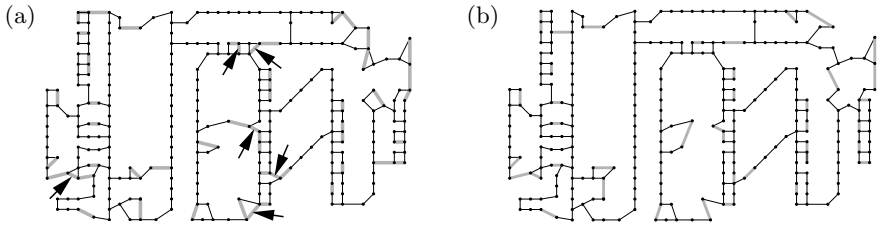

Fig. 3. Solutions to the Euclidean problem instance a280 (100) found by (a) Khuller and Thurimella's heuristic (gap: 29.4%, $|S| = 53$) and (b) the memetic algorithm (optimal, $|S| = 40$). The solutions' augmentation edges are shown in gray. In (a), the arrows mark obviously redundant edges.

spectively mutation. Furthermore, the local cost-based heuristics in the edge-selections of initialization, recombination, and mutation play a significant role.

Empirical tests indicate that the algorithm calculates solutions of high quality which are optimal in many cases. In particular the approach scales well to large problem instances due to its relatively low computational effort for the creation and local improvement of one candidate solution.

References

1. K. P. Eswaran and R. E. Tarjan. Augmentation problems. *SIAM Journal on Computing*, 5(4):653–665, 1976.
2. G. N. Frederickson and J. Jájá. Approximation algorithms for several graph augmentation problems. *SIAM Journal on Computing*, 10(2):270–283, 1981.
3. H. N. Gabow, Z. Galil, T. Spencer, and R. E. Tarjan. Efficient algorithms for finding minimum spanning trees in undirected and directed graphs. *Combinatorica*, 6(2):109–122, 1986.
4. T.-S. Hsu and V. Ramachandran. On finding a minimum augmentation to biconnect a graph. *SIAM Journal on Computing*, pages 889–912, 1993.
5. S. Khuller and R. Thurimella. Approximation algorithms for graph augmentation. *Journal of Algorithms*, 14(2):214–225, 1993.
6. I. Ljubić and J. Kratica. A genetic algorithm for the biconnectivity augmentation problem. In C. Fonseca, J.-H. Kim, and A. Smith, editors, *Proceedings of the 2000 IEEE Congress on Evolutionary Computation*, pages 89–96. IEEE Press, 2000.
7. I. Ljubić and G. R. Raidl. An evolutionary algorithm with hill-climbing for the edge-biconnectivity augmentation problem. In E. J. Boers, S. Cagnoni, J. Gottlieb, E. Hart, P. L. Lanzi, G. R. Raidl, R. E. Smith, and H. Tijink, editors, *Applications of Evolutionary Computation*, volume 2037 of *LNCS*, pages 20–29. Springer, 2001.
8. P. Moscato. Memetic algorithms: A short introduction. In D. Corne et al., editors, *New Ideas in Optimization*, pages 219–234. McGraw Hill, 1999.
9. G. R. Raidl and I. Ljubić. Evolutionary local search for the edge-biconnectivity augmentation problem. *to appear in Information Processing Letters*, 2001.
10. A. Zhu, S. Khuller, and B. Raghavachari. A uniform framework for approximating weighted connectivity problems. In *Proceedings of the 10th ACM-SIAM Symposium on Discrete Algorithms*, pages 937–938, 1999.