TECHNISCHE UNIVERSITÄT WIEN
Institut für Computergraphik und Algorithmen

Evolutionary Local Search for the Edge-Biconnectibity Augmentation Problem

G. R. Raidl, I. Ljubić

Forschungsbericht / Technical Report

TR-186-1-02-02

August 2001

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

Evolutionary local search for the edge-biconnectivity augmentation problem*

Günther R. Raidl[a†] Ivana Ljubić[a]

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

This paper considers the problem of augmenting a given graph by a cheapest possible set of additional edges in order to make the graph edge-biconnected. An application is the extension of an existing communication network to become robust against single link-failures. An evolutionary algorithm (EA) is presented which includes an effective preprocessing of the problem data and a local improvement procedure that is applied during initialization, recombination, and mutation. In this way, the EA searches the space of feasible, locally optimal solutions only. The variation operators were designed with particular emphasis on low computational effort and strong locality. Empirical results indicate the superiority of the new approach over two previous heuristic methods.

Keywords: combinatorial problems, graph algorithms, evolutionary algorithms, interconnection networks, connectivity augmentation problems

1. Introduction

In the design of communication networks, robustness against failure is an important issue. Redundant communication routes are often introduced in order to ensure that any two nodes do not loose connection in case of a single failure in a link or relaying node. An undirected graph that might represent a communication network is said to be *edge-biconnected* if at least two edges need to be removed in order to separate the graph into disconnected components. In a graph that is not edge-biconnected, a critical edge whose removal would disconnect the graph is called *uncovered edge* or *bridge*.

This article focuses on the *edge-biconnectivity augmentation* (E2AUG) problem, which is sometimes also called bridge-connectivity augmentation problem [4]. In it, an undirected, connected graph $G = (V, E)$ with node set V and edge-set E and an additional, disjoint set A of edges between nodes in V are given. Each edge $e \in A$ has associated costs $c(e) > 0$ and can be used to augment graph G. The objective is to identify a subset $S \subseteq A$ of edges with minimum total costs $c(S) = \sum_{e \in S} c(e)$ such that the augmented graph $G_S = (V, E \cup S)$ is edge-biconnected. We presuppose the fully augmented graph $G_A = (V, E \cup A)$ to be edge-biconnected, since otherwise, no feasible solution is possible.

Besides in communication network design, this problem also appears in VLSI floorplanning [15]. A general survey on related graph-connectivity problems and algorithms is given in [7]. Eswaran and Tarjan [3] proposed a polynomial-time algorithm for the special case of the E2AUG problem where all edge costs are equal and G_A is a complete graph. However, for the general case Eswaran and Tarjan showed the E2AUG problem to be *NP*-complete. This even holds when the edge costs are chosen from $\{1, 2\}$ only.

The following section gives an overview of previous heuristic methods for the E2AUG problem. A new evolutionary approach that includes an effective preprocessing of problem data and searches the domain of feasible, locally optimal solutions only is then proposed in sections 3 and 4. Initialization, recombination, and mutation operators are specifically designed for the considered problem. When deriving new offspring so-

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

†Corresponding author.
E-mail addresses: raidl@ads.tuwien.ac.at (G. Raidl), ljubic@ads.tuwien.ac.at (I. Ljubic).

lutions, these operators preserve a great amount of parental structures, i.e. the locality is strong. Furthermore, the operators' average computational effort is low, which allows a fast execution of the EA also on large graphs. Section 5 presents experimental results indicating the efficiency of this approach and its superiority over two previous heuristic methods.

2. Previous work

As described by Frederickson and Jájá [4], the problem of augmenting a general connected graph G can be effectively reduced into the problem of augmenting a spanning tree by shrinking the node sets of all already edge-biconnected components into corresponding single new nodes, Fig. 1(a) shows an example. Edges in E and A between nodes of the same edge-biconnected component are discarded, and among the edges in A connecting the same pair of components only the minimum cost edge is retained. In the example, edge $e2$ is therefore removed.

Further, Frederickson and Jájá [4] proposed an approximation algorithm which is guaranteed to give a solution with total costs being at most twice the costs of the optimum solution. This algorithm directs all edges in the spanning tree G towards a chosen root node and adds for each edge in A two corresponding oppositely directed edges. A minimum cost branching algorithm [5] is then used to find a directed spanning tree out of the root. The set of augmenting edges $S \subseteq A$ can finally be derived from this tree.

Subsequently, this algorithm was improved by Khuller and Thurimella [8] with regards to the time-complexity and recently by Zhu et al. [16, 17]. Although the latter approach has the same worst-case approximation factor of two, it gives in practice significantly better results than the previous methods. Instead of deriving all augmenting edges from a single minimum cost branching, Zhu et al. use an iterative process which fixes only one augmenting edge at a time based on some heuristic measurement of how useful each edge is. After fixing one edge, its costs are reduced to zero and a new minimum cost branching is derived. This process continues until the minimum cost branching contains only edges with zero costs and a complete set of augmenting edges is obtained.

Evolutionary algorithms (EAs) are known to be robust optimization techniques that have already been successfully applied to several hard network design problems. Most previous works in this domain deal with spanning tree structures, e.g. [9,14], or Steiner trees, e.g. [2].

Recently, we described a straight-forward hybrid EA for the E2AUG problem [11]. In this approach, a candidate solution is represented by a binary vector of length $|A|$. Each bit is associated with an edge in A and indicates whether the edge is used for augmentation, i.e. the edge is contained in S, or not. Candidates of an initial population are created by independently setting each bit to a random value. New offspring solutions are derived from pairs of parental solutions by means of traditional uniform crossover and bit-flip mutation [6]. In this way, generated candidates can be infeasible, i.e. the corresponding augmented graphs G_S may not be edge-biconnected. Such solutions are transformed into feasible ones by applying a greedy repair strategy that identifies bridges and covers them one by one by adding the cheapest appropriate edges. On a set of differently structured test problem instances, this hybrid EA obtained in most cases significantly better final solutions than the iterative method from Zhu et al. [17].

Nevertheless, this EA has also disadvantages: Since an encoded solution has always size $|A|$, the variation operators, repair, and evaluation becomes computationally expensive for larger complete or very dense graphs G_A. The required space and time is in this case at least $O(|V|^2)$, while an optimal set of augmenting edges consists always of less than $|V|$ edges according to Mader's theorem [12]. Furthermore, many different solutions created by recombination and/or mutation are mapped to one and the same feasible solution by the repair operator. This effect decreases efficiency and endangers the EA to converge too quickly to suboptimal solutions.

In the next two sections, we present a new evolutionary local search approach which overcomes these disadvantages by using a compact edge set representation, problem specific variation oper-

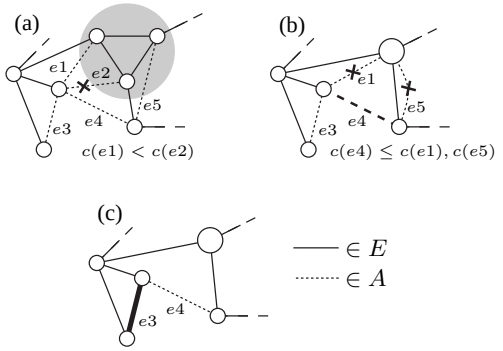

Figure 1. Preprocessing: (a) shrinking, (b) edge elimination, and (c) fixing of edges.

procedure EA:
create random initial population P of solutions;
do
  select $S_1, S_2 \in P$ via tournament selection;
  create S_{child} via recombination of S_1, S_2;
  mutate S_{child};
  if $S_{\text{child}} \notin P$ **then**
    replace worst solution in P by S_{child};
until no new best solution for Ω iterations;

Figure 2. The steady-state EA framework.

ators, and a stochastic local improvement algorithm. Preliminary results of this method are also published in [10].

3. Preprocessing

In the new evolutionary local search approach, an extended preprocessing is performed which applies the following three rules to the graph G and the additional edge set A in order to reduce $|A|$ and therefore the actual search space size.

Shrinking: All maximal edge-biconnected components of G are identified and shrinked into single new super-nodes as described in [4]. Among the edges from A that connect the same components, only the cheapest ones are retained. Edges connecting nodes of the same component are always discarded. After shrinking, G is always a tree whose edges are the bridges of the original graph. Note that now, E and A may not be disjoint anymore, and G_S and G_A are multi-graphs that may contain up to two edges for any pair of nodes. Fig. 1(a) shows an example in which the shaded subgraph will be shrinked into a super-node and $e2$ will be discarded.

Edge elimination: Let for each edge $e \in A$ the set $Q(e) \subseteq E$ be the set of all fixed edges that are *covered* by e, i.e. the path in G connecting the end-nodes of e. If there are two edges $e, e' \in A$ with $Q(e) \subset Q(e')$ and $c(e) \geq c(e')$, then e is obsolete and can be removed forever from A. Figure 1(b) shows an example in which edges $e1$ and $e5$ are obsolete due to edge $e4$. All such obsolete edges can be identified in $O(|V|^2)$ time as a byproduct from a dynamic programming algorithm that computes distance values needed for the approximation algorithm from Frederickson and Jájá [4, pp. 276–277].

Fixing of edges: We determine for each fixed edge $e_0 \in E$ the set of edges that cover e_0: $Cov(e_0) = \{e \in A \mid e_0 \in Q(e)\}$. Each edge $e \in A$ that appears as the only element in any set $Cov(e_0)$ must be contained in any feasible solution in order to cover e_0 and is therefore fixed forever by moving it from A to E; see Fig. 1(c) where edge $e3$ could be fixed.

The fixing of edges introduces new edge-biconnected components into G, which are further shrinked into new super-nodes. This shrinking, however, might enable further edge eliminations and/or a further fixing of edges. Therefore, all three parts of preprocessing are repeated until no more edges can be fixed.

4. Evolutionary local search

Although preprocessing can reduce $|A|$ significantly, it is in general not able to solve the problem completely. The steady-state EA framework

[1] shown in Fig. 2 is applied for further optimization.

Candidate solutions are represented by directly storing the set of selected edges S in form of a hash-table. In this way, only $O(|S|)$ space is needed, where $|S|$ is always less than $|V|$ in our case, and an edge can be added or deleted in constant time.

In each iteration, one new offspring solution is created by selecting two parents S_1 and S_2 and performing recombination and mutation. The offspring always replaces the worst solution in the population with one exception: A new solution that is a duplicate of a solution already contained in the population is discarded to ensure a minimum degree of diversity and, thus, to counteract premature convergence.

The tournament selection used for obtaining a parent chooses a group of k solutions from the population independently at random (multiple choices of the same solution are allowed), and the solution with the smallest total costs $c(S)$ is the selected one.

4.1. Stochastic local improvement

As a central element of the EA, a stochastic local improvement procedure is used during initialization, recombination, and mutation. In this way, the proposed EA searches the space of locally optimal solutions only; compare memetic algorithms [13], which follow a similar principle.

Our stochastic local improvement removes redundant edges from a feasible edge set S in a random way until S becomes locally edge-minimal, i.e. no further edges can be removed without making S infeasible by introducing bridges into G_S.

Figure 3 shows the detailed algorithm of local improvement. It starts by determining for each edge $e_0 \in E$ the number $n_{\text{cov}}(e_0)$ of edges in S that cover it. The needed path $Q(e) \subseteq E$ of fixed tree edges that are covered by edge e can be determined efficiently in $O(|Q(e)|)$ time if during preprocessing a complete depth-first search starting from an arbitrary root node is once performed on the tree G and each node's predecessor and depth are stored. In the second part, local improvement checks each edge $e \in S$ in a random order, if the edge is for some fixed edge $e_0 \in Q(e)$ the only

procedure locally improve S:
$\forall e_0 \in E : n_{\text{cov}}(e_0) \leftarrow 0$;
for each $e \in S$ **do**
  $\forall e_0 \in Q(e) : n_{\text{cov}}(e_0) \leftarrow n_{\text{cov}}(e_0) + 1$;
$T \leftarrow S$;
do
  select $e \in T$ via tournament selection
    (prefer more expensive edges);
  $T \leftarrow T \setminus \{e\}$;
  if $\forall e_0 \in Q(e) : n_{\text{cov}}(e_0) \geq 2$ **then**
    $S \leftarrow S \setminus \{e\}$;
    $\forall e_0 \in Q(e) : n_{\text{cov}}(e_0) \leftarrow n_{\text{cov}}(e_0) - 1$;
while $T \neq \emptyset$.

Figure 3. Stochastic local improvement.

cover ($n_{\text{cov}}(e_0) = 1$). If this is not the case, e is considered redundant and removed from S; n_{cov} is updated accordingly.

Note that the order of checking the edges in S is crucial. Edges processed first will in general be removed with higher probability. Since we are interested in making solutions as cheap as possible, we heuristically bias the ordering towards more expensive edges coming first: The next edge to be processed is always chosen by performing a tournament selection on all yet unprocessed edges in S, i.e. from a group of k_{locimp} randomly chosen edges, the most expensive edge is selected.

The average computational effort of the complete procedure is $O(|S| \log |V|)$ since $|Q(e)| = O(\log |V|)$ in average.

4.2. Initialization

Local improvement can immediately be used to generate the EA's initial solutions by calling it with $S = A$. Enough diversity is provided in such an initial population due to the randomness of the edge processing order as long as k_{locimp} is not too large.

4.3. Recombination

Recombination is performed by applying local improvement to the union $S_1 \cup S_2$ of both parents' edge sets. In this way, a new feasible and locally optimal offspring is always created containing only edges that already appeared in

procedure mutate S:
do n_{mut} times:
  choose $e \in S$ randomly;
  $S \leftarrow S \setminus \{e\}$;
  $\forall e_0 \in Q(e) : n_{\mathrm{cov}}(e_0) \leftarrow n_{\mathrm{cov}}(e_0) - 1$;
  for each $e_0 \in Q(e)$ s.t. $n_{\mathrm{cov}}(e_0) = 0$
      in random order **do**
    select $e' \in Cov(e_0) \setminus \{e\}$
        (prefer cheaper edges);
    $S \leftarrow S \cup \{e'\}$;
    $\forall e'_0 \in Q(e') : n_{\mathrm{cov}}(e'_0) \leftarrow n_{\mathrm{cov}}(e'_0) + 1$;
  locally improve S; **end**.

Figure 4. Mutation.

one of the parental solutions. Thus, meaningful building-blocks [6] can effectively be transported from parents to offsprings, and strong locality is provided. Since joining the parental edge sets takes $O(|S_1|+|S_2|)$ time and $|S| < |V|$ for any locally improved solution S due to Mader's theorem [12], the overall computational effort of recombination is $O(|V| \log |V|)$ in average.

4.4. Mutation

The purpose of mutation in the proposed EA is to vary solutions slightly. It counteracts a premature convergence by introducing edges from A that are not contained in any solution of the population. Figure 4 shows the detailed algorithm.

The procedure chooses an edge $e \in S$ randomly and removes it. Each fixed edge $e_0 \in Q(e)$ which is now uncovered ($n_{\mathrm{cov}}(e_0) = 0$) is determined and covered anew by adding a replacement edge from $Cov(e_0)$ (the set of all edges from A covering e_0, see section 3). Since edges from S might become redundant when adding new replacement edges, local improvement is again performed as final step. The whole mutation is repeated n_{mut} times, where n_{mut} is a strategy parameter controlling how disruptive mutation is.

As an additional heuristic, the random choice of a replacement edge from $Cov(e_0)$ is biased towards cheaper edges by using tournament selection again.

Provided that $Cov(e_0)$ is determined for all $e_0 \in E$ already during preprocessing, the average computational effort of mutation is again $O(|V| \log |V|)$ due to the final local improvement.

5. Empirical results

We present empirical results for the most difficult problem instances from [11], which have been created using a generator from Zhu [16], and some new, larger instances. Since shrinking can always effectively reduce the problem of augmenting a general connected graph G to the problem of augmenting a tree, G is always a spanning tree in these test instances and $|E| = |V| - 1$.

Table 1 shows properties of the considered instances. All graphs were randomly created. For instances A3 to R2, costs of all edges $e \in A$ were randomly chosen integers from the intervals printed in column $c(e)$. Instances E1 to E3 are Euclidean problems where nodes are randomly placed points in a square region of the plane, and edge costs correspond to the points' Euclidean distances. G is a random spanning tree in case of instances A3 to R2 and a minimum-cost spanning tree in the Euclidean cases.

Furthermore, Table 1 shows the impacts of preprocessing. Especially when the fully augmented graph G_A is sparse, as in instances A3, B1, and B6, the fixing of edges together with the iterative shrinking can dramatically reduce the number of nodes and therefore the problem size. In this case, or when G is similar to a star, edge elimination is usually less effective. On the other hand, if G_A is dense as in the remaining instances, none or only few edges can typically be fixed and no substantial shrinking is therefore possible, but the edge elimination step reduces A often dramatically.

In average, preprocessing could shrink A to about the half of its original size. Especially for larger problem instances, times t_{pre} for preprocessing are neglectable in comparison to the EA's total execution times.

The following EA setup was used since it proved to work well in most situations according to extensive preliminary tests. Population size: $|P|$=100; group size for main tournament selection: $k = 5$, for edge-selection during local improvement: $k_{\mathrm{locimp}} = 5$, and for edge-selection

Table 1
Problem instances and results of preprocessing.

Instance	$\|V\|$	$\|A\|$	$c(e)$	$\|V_{\text{pre}}\|$	$\|A_{\text{pre}}\|$	t_{pre} [s]
A3	40	29	[1,780]	12	13	0.1
B1	60	55	[1,1770]	8	4	0.1
B6	70	81	[1,2415]	31	39	0.2
D3	90	366	[1, 4005]	90	278	0.3
D5	100	398	[1,4950]	100	301	0.4
M1	70	290	[1,2415]	70	227	0.2
N1	100	1104	[10,50]	100	687	0.6
N2	110	1161	[10,50]	110	734	0.6
R1	200	9715	[1,100]	200	3995	11.2
R2	200	9745	[5,100]	200	3702	10.9
E1	200	19701	Euclidean	200	4104	25.8
E2	300	11015	Euclidean	300	4462	31.5
E3	400	7621	Euclidean	400	4806	51.6

during mutation: $k_{\text{mut}} = 4$; number of edges removed during mutation: $n_{\text{mut}} = 5$. A run of the EA was terminated when no new best solution had been found during the last $\Omega = 10,000$ iterations.

Table 2 shows results of the iterative heuristic (ITH) from Zhu et al. (reprinted from [16]), our previous hybrid EA (HEA) [11], and the new evolutionary local search (ELS). For each problem instance, the total costs $c(S^*)$ of the best known solution S^* are listed in the table. Qualities of solutions S obtained by the three approaches are printed as percentage *gaps* with respect to these best known solutions: $gap = (c(S) - c(S^*))/c(S^*) \cdot 100\%$. Results were obtained by averaging over 100 runs per instance in case of ELS and 10 runs per instance in case of HEA. ITH gives always the same solutions due to its determinism. The success rate *sr* denotes in how many of the performed runs a solution having costs $c(S^*)$ could be identified. It is remarkable that ELS was able to find best known solutions S^* in all 100 runs for instances A3 to M1, N2, and R1. ELS's success rates and average gaps are for all considered instances significantly better than those of HEA and ITH, with the exception of A3, B1, B6, and M1 for which HEA could also identify best known solutions in all runs.

For HEA and ELS, Table 2 shows also average numbers of created and evaluated candidate solutions *evals* and CPU-times t (in seconds) until the finally best solutions had been obtained. ELS needed always significantly fewer evaluations and less time than HEA. For E1, the instance with the largest set A, ELS was more than 100 times faster than HEA.

6. Conclusions

The main features of the proposed evolutionary local search for the E2AUG problem are: (a) the deterministic preprocessing of problem data, which is able to reduce the number of edges under question dramatically, (b) the stochastic local improvement with its cost-based heuristic, and (c) the specifically designed recombination and mutation operators which are based on local improvement and provide strong locality. Because of these, the EA focuses on the hard core of the problem and efficiently searches the space of feasible, locally optimal solutions only. Due to the relatively low computational effort of $O(|V| \log |V|)$ for recombination and mutation, the EA scales well to larger problem instances.

Future work will include a generalization of the approach for k-edge connectivity and vertex connectivity augmentation.

Table 2
Average results of ITH, HEA, and ELS.

Instance	$c(S^*)$	ITH	HEA (10 runs)				ELS (100 runs)			
		gap	*gap*	*sr*	*evals*	t [s]	*gap*	*sr*	*evals*	t [s]
A3	6607.0	2.98%	0.00%	100%	380	0.1	0.00%	100%	0	0.1
B1	15512.0	0.00%	0.00%	100%	50	0.1	0.00%	100%	0	0.1
B6	19022.0	0.02%	0.00%	100%	9415	4.5	0.00%	100%	7	0.2
D3	20321.0	0.44%	0.15%	60%	4130	7.5	0.00%	100%	607	2.6
D5	19355.0	1.28%	0.35%	80%	18160	32.2	0.00%	100%	499	2.7
M1	2940.0	2.38%	0.00%	100%	15405	14.7	0.00%	100%	181	1.2
N1	383.0	3.92%	2.61%	10%	95335	230.4	0.47%	42%	3998	10.4
N2	429.0	4.90%	2.38%	0%	120385	545.0	0.00%	100%	3793	11.3
R1	121.4	–	1.12%	40%	244325	12398.3	0.00%	100%	12410	135.3
R2	320.5	–	6.73%	0%	243085	11434.4	0.67%	13%	38912	218.5
E1	2873.8	–	12.84%	0%	236305	20740.1	0.98%	23%	34129	191.0
E2	9355.2	–	8.93%	0%	236480	22602.5	0.42%	2%	97764	731.0
E3	21329.1	–	8.38%	0%	246640	23970.4	0.55%	2%	113831	1451.4

REFERENCES

1. T. Bäck, D. B. Fogel, and Z. Michalewicz. *Handbook of Evolutionary Computation*. Oxford University Press, New York, 1997.
2. H. Esbensen. Finding (near-)optimal Steiner trees in large graphs. In L. J. Eshelman, editor, *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 485–491. Morgan Kaufmann, 1995.
3. K. P. Eswaran and R. E. Tarjan. Augmentation problems. *SIAM Journal on Computing*, 5(4):653–665, 1976.
4. G. N. Frederickson and J. Jájá. Approximation algorithms for several graph augmentation problems. *SIAM Journal on Computing*, 10(2):270–283, 1981.
5. 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.
6. D. E. Goldberg. *Genetic Algorithms in Search, Optimization, and Learning*. Addison-Wesley, Reading, Massachusetts, 1989.
7. S. Khuller. Approximation algorithms for finding highly connected subgraphs. In D. Hochbaum, editor, *Approximation Algorithms for* NP*-hard Problems*, pages 236–265. PWS Publishing, Boston, MA, 1996.
8. S. Khuller and R. Thurimella. Approximation algorithms for graph augmentation. *Journal of Algorithms*, 14(2):214–225, 1993.
9. J. Knowles and D. Corne. A new evolutionary approach to the degree constrained minimum spanning tree problem. *IEEE Transactions on Evolutionary Computation*, 4(2):125–134, 2000.
10. I. Ljubić and G. R. Raidl. An evolutionary algorithm with stochastic hill-climbing for the edge-biconnectivity augmentation problem. In E. Boers, J. Gottlieb, P. Lanzi, R. Smith, S. Cagnoni, E. Hart, G. Raidl, and H. Tijink, editors, *Applications of Evolutionary Computation*, volume 2037 of *LNCS*, pages 20–29. Springer, 2001.
11. I. Ljubić, G. R. Raidl, and J. Kratica. A hybrid GA for the edge-biconnectivity augmentation problem. In K. Deb, G. Rudolph, X. Yao, and H.-P. Schwefel, editors, *Proceedings of the 2000 Parallel Problem Solving from Nature VI Conference*, volume 1917 of *LNCS*, pages 641–650. Springer, 2000.
12. W. Mader. Minimale n-fach kantenzusammenhängende Graphen. *Math. Ann.*, 191:21–

28, 1971.
13. P. Moscato. Memetic algorithms: A short introduction. In D. C. et al., editor, *New Ideas in Optimization*, pages 219–234. McGraw Hill, Berkshire, England, 1999.
14. C. C. Palmer and A. Kershenbaum. Representing trees in genetic algorithms. In D. Schaffer, H.-P. Schwefel, and D. B. Fogel, editors, *Proceedings of the First IEEE Conference on Evolutionary Computation*, pages 379–384. IEEE Press, 1994.
15. S. Tsukiyama, K. Kioke, and I. Shirakawa. An algorithm to eliminate all complex triangles in a maximal planar graph for use in VLSI floorplanning. In M. Sarrafzadeh and D. T. Lee, editors, *Algorithmic Aspects of VLSI Layout*. World Scientific Publishing, 1993.
16. A. Zhu. A uniform framework for approximating weighted connectivity problems. B.Sc. thesis, University of Maryland, MD, May 1999.
17. 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.