An Evolutionary Algorithm with Stochastic Hill-Climbing for the Edge-Biconnectivity Augmentation Problem

Ivana Ljubić and Günther R. Raidl

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

Abstract. Augmenting an existing network with additional links to achieve higher robustness and survivability plays an important role in network design. We consider the problem of augmenting a network with links of minimum total cost in order to make it edge-biconnected, i.e. the failure of a single link will never disconnect any two nodes. A new evolutionary algorithm is proposed that works directly on the set of additional links of a candidate solution. Problem-specific initialization, recombination, and mutation operators use a stochastic hill-climbing procedure. With low computational effort, only locally optimal, feasible candidate solutions are produced. Experimental results are significantly better than those of a previous genetic algorithm concerning final solutions' qualities and especially execution times.

1 Introduction

All communication networks are designed for certain demands and requirements. In the course of time, traffic demands typically increase and the networks are often not as satisfying as at the beginning. Therefore, *the augmentation of networks* by additional links plays an important role in network design. In addition to the increase of band-width, an increase of robustness and survivability is often needed. A network can be made robust against failures in connections between two sites or against site failures. The costs of such augmentations should usually be as small as possible.

The robustness of a certain network, is in graph theory described by the *vertex* and *edge k-connectivity*. A connected undirected graph $G = (V, E)$ has edge (vertex) connectivity k if at least k edges (vertices) must be deleted to disconnect G. The removal of vertices hereby includes the removal of all adjacent edges. Therefore, a vertex k-connected network is always edge k-connected, but not necessarily vice versa.

The problem of augmenting a graph to become $k = 1$ connected is identical to the *minimum spanning tree* (MST) problem, which can be solved efficiently in polynomial time. However, in practical communication networks, a larger connectivity-level for higher reliability is often needed. From the other side, costs usually limit the connectivity level k to a small value.

In this paper, we concentrate on edge-connectivity with $k = 2$. This problem is also called *edge biconnectivity augmentation problem* (E2AUG). Our goal is, therefore, to augment a given network with additional links of minimum total costs, to make it edge-biconnected.

Eswaran and Tarjan [1] showed that E2AUG is $\mathcal{NP}$-hard. The problem remains $\mathcal{NP}$-hard, even in the case when all connections have weights chosen from the set $\{1, 2\}$ only. Due to the hardness of the problem, it was addressed by heuristic methods including a hybrid genetic algorithm (HGA) [7]. In contrast to this previous HGA, we present here a new evolutionary algorithm (EA) based on a powerful preprocessing and a straight-forward edge-set representation. The recombination and mutation operators produce only feasible solution candidates and contain a local stochastic hill-climbing which removes redundant edges. That way, the EA searches the space of locally optimal solutions only.

The initialization, recombination, and mutation operators are specifically designed for the considered problem. Recombination and mutation preserve a great amount of parental structures, i.e. the locality is high. The average computational effort of recombination and mutation operators is low, which allows a fast execution on large graphs, too. Empirical results indicate the new EA is significantly better concerning the quality of final solutions, as well as the execution times when compared to the previous HGA and another iterative heuristic for the E2AUG.

The next section provides a mathematical definition of E2AUG and a summary of previous work related to the problem. In Section 3, an explanation of the preprocessing is given. Section 4 describes the EA with its stochastic local hill-climbing procedure in detail. An empirical comparison to the previous HGA is given in Section 5, and final conclusions are drawn in Section 6.

2 The Edge-Biconnectivity Augmentation Problem

Given are a connected, undirected graph $G = (V, E)$, and an additional set AUG of edges connecting nodes in V ($AUG \cap E = \emptyset$). Each edge $e \in AUG$ has associated costs $c(e) > 0$. The graph $G_A = (V, E \cup AUG)$ is edge-biconnected. The goal is to augment graph G using a subset S of edges from AUG with minimum total costs $c(S) = \sum_{e \in S} c(e)$, so that graph $G_S(V, E \cup S)$ is also edge-biconnected.

In graph G, an edge $e \in E$ is called a *bridge* if its deletion disconnects G. G_S must therefore not contain any bridges. That is why this problem is also called *bridge-connectivity augmentation problem.*

The problem has been stated the first time by Eswaran and Tarjan [1]. In this work a polynomial algorithm for E2AUG is given for the specific case when all edge costs are the same and graph G_A is complete. A survey on several related problems and approximation algorithms is given by Khuller [4].

In 1981, Frederickson and Jájá [2] proposed an approximative algorithm for E2AUG based on the following steps:

Firstly, all already biconnected components in G are shrinked into "supernodes", whereby all self-loops are discarded; from multiple edges $e \in AUG$ connecting the same pair of nodes in the shrinked graph, only the cheapest edge is retained. In this way, the problem of augmenting a general connected graph G, can always be reduced to the problem of augmenting a spanning tree.

In the next step, the shrinked graph G is interpreted as a directed tree: a random root r is chosen and all edges in E are directed toward this root. After that, the algorithm searches for a minimum outgoing branching, i. e. a directed tree with paths from the root r to all other nodes (Gabow et al. [3]), using edges from the shrinked set AUG and E. A feasible set of augmenting edges $S \subset AUG$ can finally be derived from the set of edges included in this branching. It is proven that this algorithm determinates a solution with costs no greater than two times the costs of the optimal solution.

The time complexity of this algorithm has been improved by Khuller and Thurimella [5]. Recently, Zhu et al. [11, 12] proposed an iterative scheme based on the branching algorithm. They provide a heuristic formula for measuring how good a certain augmenting edge in a determined outgoing branching can be. Using this formula, only one edge at a time, as one step of an iterative process, is fixed, and the edge's cost is set to zero. Then, a new minimum outgoing branching is derived, and the edge is fixed. This process continues until the evaluated branching has zero costs and a complete set S is obtained.

Ljubic et al. [7] proposed a hybrid genetic algorithm (called HGA) for E2AUG. This algorithm is based on a binary encoding, in which each bit corresponds to an edge in AUG, on standard uniform crossover, and on bit-flip mutation. Infeasible solutions are repaired using a greedy repair-algorithm in a Lamarckian way. This algorithm temporarily removes the bridges one by one from an infeasible solution and searches for the cheapest edge from AUG connecting the two separated components. For a better performance, the algorithm uses caching [6].

Although better results are obtained by HGA than by several previous approaches, this method has also disadvantages: The genetic code has length $|AUG|$, which is not efficient for larger complete or dense graphs. The required space and time to evaluate a solution is $O(|V|^2)$ in such cases, while the number of edges included in biconnectivity augmentation is always less then $|V|$, according to Mader's theorem [8]. Furthermore, many genetic codes created by recombination and/or mutation are mapped to one and the same phenotypic solution due to the repair operator. This effect endangers the GA to converge too quickly to suboptimal solutions.

In this paper, we present a new evolutionary approach which overcomes these disadvantages using a compact edge-set encoding, problem-based operators, and a local stochastic hill-climbing. This EA searches the space of local optima only. Such an approach belongs to a broader group of combinatorial optimization algorithms, called *local-search-based memetic algorithms* [9].

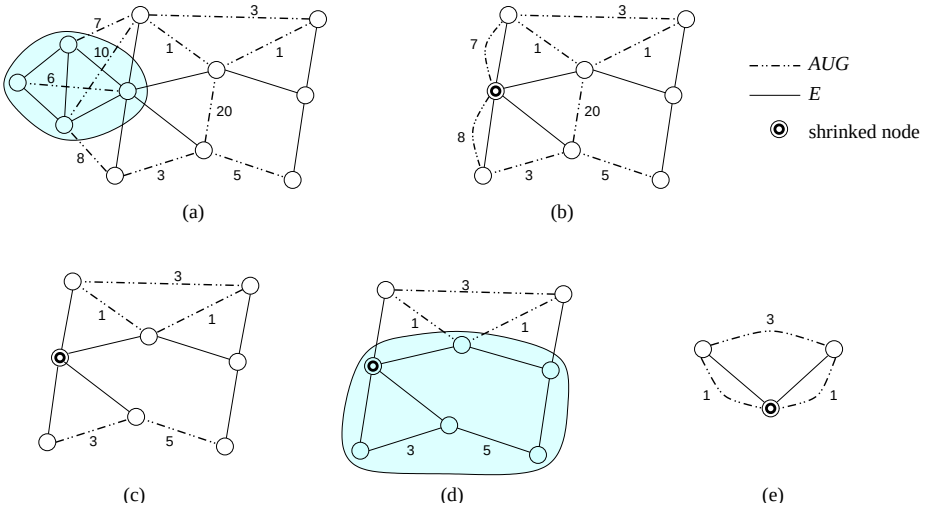

Fig. 1. An example for preprocessing: (a) given graph $G = (V, E)$ and set AUG, (b) after shrinking, (c) after elimination of edges, (d) after fixation of edges from AUG, (e) after another shrinking.

3 Preprocessing

Good preprocessing can reduce a problem's search space significantly. Our preprocessing is based on the next three steps which are illustrated in Fig. 1:

Shrinking: This reduction has been originally described by Frederickson and Jájá [2]. Edge-biconnected components of a graph are its maximal edge-biconnected subgraphs (maximal in the sense that no other node from the graph can be added, without violating biconnectivity). By this procedure, all edge-biconnected subgraphs are found and shrinked into "super-nodes". From the edges from AUG that connect the same components, only the cheapest ones are included and all others are discarded. Self-loops, i.e. augmenting edges connecting nodes of the same component, are always discarded. For each edge in the shrinked graph a reference back to the corresponding edge of the original problem is stored. After shrinking, the graph has always a tree structure, where all edges represent bridges of the initial configuration. Note that now, $G_S = (V, E \cup S)$ and $G_A = (V, E \cup AUG)$ can become multigraphs, since AUG and E are not necessarily disjoint anymore (see Fig. 1 (b)).

Edge elimination: Edge $e_0 \in E$ is *covered* by an edge $e = (u, v) \in AUG$ if e_0 lies on the tree path (in G) connecting nodes u and v. This procedure detects and removes each edge e_{in} from AUG, for which some other edge $e_{out} \in AUG$ exists, such that the tree-path (in G) covered by e_{in} is a subset of the tree path covered by e_{out} and e_{out} is cheaper then e_{in}. More formally, if $e_{in} = (u, v) \in AUG$, and:

$$Path(e) = \{e_0 \mid e_0 \in E \text{ and } e_0 \text{ is part of the path connecting } u \text{ and } v \text{ in } G\}$$

then, e_{in} is obsolete if:

$$\exists\, e_{out} = (s,t) \in AUG \text{ such that } Path(e_{in}) \subset Path(e_{out}) \wedge c(e_{in}) \geq c(e_{out}).$$

Frederickson and Jájá [2, pp. 276–277] proposed a dynamic programming algorithm for their branching based heuristic, that computes special distance values for all edges in AUG and identifies all obsolete edges in the above sense as a byproduct in $O(|V|^2)$ time.

Usually, the more star-like graph G is, the less edges can be eliminated by means of this procedure.

Edge fixation: This procedure identifies edges that must necessarily be included in the final solution. The sets $Cover(e_0),\ e_0 \in E$, are the sets of all augmenting edges $e = (u,v) \in AUG$ such that e_0 lies on the (u,v)-path in G, i.e.

$$Cover(e_0) = \{e = (u,v) \in AUG \mid e_0 \in Path(e)\}. \tag{1}$$

If there exists an edge $e_0 \in E$ such that $|Cover(e_0)| = 1$, then edge e must appear in all feasible solutions, since this is the only possibility to "cover" e_0. Edge e is therefore fixed by moving it from AUG to E. After such a fixation, a new edge-biconnected component is created, which can further be shrinked into a single new super-node. Since shrinking can enable further fixations, the process is repeated until no more edges from AUG can be fixed.

Note that the more sparse graph G_A is, the more edges can typically be fixed.

4 The Evolutionary Algorithm

Although preprocessing reduces the size of the problem, it is in general not able to solve the problem completely. Therefore, we apply the following evolutionary algorithm.

4.1 Edge-Set Encoding

Each candidate solution is directly represented by the set S of selected edges. For this purpose, we use a hashed array as data structure. Insertion as well as deletion of a single edge and the check whether an edge is contained or not take always constant time. Furthermore, the space needed to store each individual is $O(S)$, where $|S| < |V|$.

4.2 Stochastic Hill-Climbing

The central part of the new approach is the local stochastic hill-climbing procedure, incorporated in the initialization, crossover, and mutation operators. This procedure removes redundant edges from a given feasible solution in an indeterministic way until the solution becomes edge-minimal concerning the biconnectivity property. This algorithm checks each edge in S in random order if it can be removed without making the solution infeasible.

```
procedure hill-climbing(var S):
begin
    n_cov(e_0) ← 0, for each e_0 ∈ E;
    for each e ∈ S do
      for each e_0 ∈ Path(e) do
        n_cov(e_0) ← n_cov(e_0) + 1;
    while not all edges e ∈ S are processed
      select a yet unprocessed edge e ∈ S;
      if n_cov(e_0) ≥ 2, ∀e_0 ∈ Path(e)
        S ← S \ {e};
        for each e_0 ∈ Path(e)
          n_cov(e_0) ← n_cov(e_0) − 1;
end
```

Fig. 2. The EA's stochastic hill-climbing.

For each edge $e_0 \in E$ we determine the number of all edges from S that "cover" e_0 in an initial step:

$$n_{cov}(e_0) = |\{e \mid e \in S \text{ and } e_0 \in Path(e)\}|.$$

An edge $e \in S$ is then *redundant* and can be removed if:

$$\forall\, e_0 \in Path(e) \;:\; n_{cov}(e_0) \geq 2.$$

After each edge elimination from S, $n_{cov}(e_0)$ is updated accordingly.

The pseudo-code presented in Figure 2 shows the algorithm in more detail. During hill-climbing, each set $Path(e)$, $\forall e \in AUG$, can be determined in $O(|path(e)|)$ time, if a depth-first search started from an arbitrarily chosen root is performed and depth and parent informations are stored for each node [10]. Hence, the worst-case execution time needed for determining all $n_{cov}(e_0)$, $e_0 \in Path(e)$, is $O(|V||S|)$, but in average the whole algorithm runs in $O(|S| \log |V|)$ time.

4.3 Initialization

In order to create feasible initial solutions, we apply the described stochastic hill-climbing procedure to the whole set AUG of edges that can be used for augmentation, i.e. $S = AUG$. Due to the indeterminism of hill-climbing, generated solutions are in general different and enough initial diversity is provided.

4.4 Edge Crossover

When designing a suitable crossover operator, our main goal was to produce a new solution that inherits as many parental structures as possible in order to provide a high level of locality. This can be accomplished by setting a child's edge-set to the union of the parental edge-sets and applying local stochastic hill-climbing to it.

```
procedure edge-delete mutation(var S):
begin
    do p_mut times:
      choose e ∈ S randomly;
      S ← S \ {e};
      for each e_0 ∈ Path(e) do
        n_cov(e_0) ← n_cov(e_0) − 1;
      for each {e_0 | e_0 ∈ Path(e) ∧ n_cov(e_0) = 0} do
        select e_1 from Cover(e_0) \ {e} randomly;
        S ← S ∪ {e_1};
        for each e' ∈ Path(e_1) do
          n_cov(e') ← n_cov(e') + 1;
      hill-climbing(S);
end
```

Fig. 3. Edge-delete mutation.

In this way, a new locally optimal solution is efficiently created out of the parental edges only. Nevertheless, even if the same two parents are again selected for mating, a different offspring is generated with high probability.

Meaningful building-blocks will be transmitted from parents to offsprings, and strong locality is provided. Since $|S| < |V|$ for each parental edge-set, the computational effort of edge crossover is only $O(|V| \log |V|)$.

4.5 Edge-Delete Mutation

The mutation operator's main purpose is to counteract premature convergence and to maintain enough diversity in the population by introducing new edges from $AUG \setminus S$. Edge-delete mutation replaces a randomly selected edge from S by one or more different, appropriate edges from $AUG \setminus S$, so that edge-biconnectivity is maintained. Since the offspring generated in this way is not necessarily edge-minimal anymore, local stochastic hill-climbing is finally applied again. The algorithm presented in Fig. 3 shows more details.

To be able to perform the mutation efficiently, we suggest to determine the set $Cover(e_0)$ as defined in (1) for each $e_0 \in E$ as a part of preprocessing. Then the mutation operator needs only $O(|V| \log |V|)$ time, too.

The parameter p_{mut} is the number of times an edge is substituted and therefore controls how strong mutation will actually change a certain solution.

4.6 Edge-Cost Based Heuristics

Usually, cheaper edges will appear more frequently in optimal solutions than expensive edges. Based on this observation, we include additional cost-based heuristic in the hill-climbing and mutation algorithms.

Heuristic in hill-climbing: During hill-climbing, the order of processing the edges in S is crucial. In each iteration, we bias the selection of the edge coming next towards more expensive edges by performing a tournament selection on all yet unprocessed edges in S. From a group of k_{impr} randomly chosen edges, the most expensive edge is selected.

Heuristic in edge-delete mutation: For mutation we select each replacement-edge (needed to cover a newly introduced bridge e_0) from $Cover(e_0) \setminus \{e\}$ by using a tournament selection. Now, cheaper edges need to be preferred; thus, the edge with the smallest costs is selected from a randomly chosen group of size k_{mut}.

4.7 General EA Properties

We used a steady-state evolutionary algorithm in which only one new solution is created by means of crossover and mutation. The new solution always replaces the worst solution with one exception: Only new solutions that are not duplicates of solutions already in population are accepted in order to maintain higher diversity. Parents are chosen using tournament selection with the group size k.

5 Experimental Results

In this section we present experimental results of the proposed EA with stochastic hill-climbing (EASHC) and the previous hybrid genetic algorithm (HGA) from [7].

Since the problem of augmenting a general connected graph G, can effectively be reduced to augmenting a tree (see Sec. 3), G is always a spanning tree in our test instances. Table 1 shows the main properties of the considered instances. Columns $|V_{pre}|$ and $|AUG_{pre}|$ indicate the numbers of nodes and the numbers of augmenting edges of G, respectively, after performing preprocessing. Instances A3 to N2 were created using a generator from Zhu et al. [12]. All graphs were randomly generated; column $c(e)$ shows the intervals from which costs for each $e \in AUG$ were chosen. Results for problem instances A3 to N2 were adopted from [7]. Instances R1 to E3 are new and larger than all previously tested ones. In contrast to the other instances, E1 to E3 are Euclidean problems in which nodes represent randomly placed points in the plane and edge costs correspond to the points' Euclidean distances.

It can further be observed that especially when G_A is sparse (as in instances A_3, B_1, B_6), the fixing of edges together with the new iterative shrinking and edge-elimination can dramatically reduce the problem size. On the other hand, if G_A is dense (instances N1 to E3), edge-fixation is not able to reduce the number of nodes, but the edge-elimination is more effective. Especially for larger problem instances, preprocessing times t_{pre} are neglectable in comparison to the EA's total execution times, see Table 2. Preprocessing was able to reduce the problem size $|AUG|$ significantly: in case of instance B1 to $\sim 1/7$, in average to about the half.

Table 1. Properties of considered problem instances.

instance	$\|V\|$	$\|AUG\|$	$c(e)$ in	$\|V_{pre}\|$	$\|AUG_{pre}\|$	$t_{pre}[s]$
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
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

Table 2. Average results of HGA and EASHC.

		HGA				EASHC			
inst.	*best*	*gap*[%]	$\sigma(gap)$	*eval*	$t[s]$	*gap*[%]	$\sigma(gap)$	*eval*	$t[s]$
A3	6607.0	0.0	0.0	380	0.1	0.0	0.0	0	0.1
B1	15512.0	0.0	0.0	50	0.0	0.0	0.0	0	0.1
B6	19022.0	0.0	0.0	9400	4.5	0.0	0.0	7	0.2
N1	383.0	2.6	2.0	95350	230.4	0.5	0.4	3998	10.4
N2	429.0	2.3	1.3	120400	544.9	0.0	0.0	3793	11.3
R1	121.4	1.1	1.0	244325	12398.3	0.0	0.1	12410	135.3
R2	320.5	6.7	1.7	243085	11434.4	0.7	0.1	38912	218.5
E1	2873.8	12.8	5.6	236305	20740.1	1.0	0.9	34129	191.0
E2	9355.2	8.9	1.7	236480	22602.5	0.4	1.6	97764	731.0
E3	21329.1	8.4	2.0	246640	23970.4	0.5	1.3	113831	1451.4

Suitable EA parameters were determined by extensive preliminary tests: population size $|P| = 100$, group size for tournament selection of the EA $k = 5$, during recombination $k_{impr} = 5$, and during mutation $k_{mut} = 4$. The mutation rate is $p_{mut} = 5$. Each run was terminated when no new best solution was found within the last 100,000 created solutions.

Table 2 presents average results obtained from 100 runs/instance in case of EASHC, and 10 runs/instance in case of HGA. Column *best* shows each problem instance's best known solution. For both algorithms percentage *gaps* of the final solutions' average total costs to the best known values *best* and standard deviations of gaps $\sigma(gap)$ are given. *Evals* indicates the average number of evaluated solutions until the finally best solution had been obtained, and t is the corresponding execution time in seconds.

For all problem instances EASHC's final solutions are better than or at least equally good as those obtained by HGA. Nevertheless, EASHC needs in all cases significantly fewer iterations and was much faster. For instance E1, EASHC was more than 100 times faster.

6 Conclusion

The proposed EA has the following advantages: initialization, recombination, and mutation produce only feasible, locally optimal solution candidates, due to the local stochastic hill-climbing. The iterative process is computationally efficient since crossover and mutation run in $O(|V| \log |V|)$ time. Therefore, the approach scales well to larger problem instances.

The direct representation in combination with the proposed variation operators provides strong locality. In particular, crossover always generates new solutions out of inherited parental edges only. Other investigations indicate that the cost-based heuristics included in hill-climbing and mutation increase the performance of the proposed EA substantially. The proposed EA obtained better final solutions in dramatically shorter execution times than the previous hybrid genetic algorithm.

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

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. S. Khuller, B. Raghavachari, and N. Young. Low-degree spanning trees of small weight. *SIAM Journal of Computing*, 25(2):355–368, 1996.
5. S. Khuller and R. Thurimella. Approximation algorithms for graph augmentation. *Journal of Algorithms*, 14(2):214–225, 1993.
6. J. Kratica. Improving performances of the genetic algorithm by caching. *Computers and Artificial Intelligence*, 18(3):271–283, 1999.
7. 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.
8. W. Mader. Minimale n-fach kantenzusammenhängende Graphen. *Math. Ann.*, 191:21–28, 1971.
9. 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.
10. R. E. Tarjan. Depth first search and linear graph algorithms. *SIAM Journal of Computing*, 1:146–160, 1972.
11. A. Zhu. A uniform framework for approximating weighted connectivity problems. B.Sc. thesis, University of Maryland, MD, May 1999.
12. 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.