A Hybrid GA for the Edge-Biconnectivity Augmentation Problem

Ivana Ljubić[1], Günther R. Raidl[1], and Jozef Kratica[2]

[1] Institute for Computer Graphics, Vienna University of Technology,
Favoritenstraße 9–11/186, A-1040 Vienna, Austria
{ljubic|raidl}@apm.tuwien.ac.at
[2] Faculty of Mechanical Engineering
Strumicka 92/5, Belgrade, Serbia
jkratica@matf.bg.ac.yu

Abstract. In the design of communication networks, robustness against failures in single links or nodes is an important issue. This paper proposes a new approach for the $\mathcal{NP}$-complete edge-biconnectivity augmentation (E2AUG) problem, in which a given graph $G_0(V, E_0)$ needs to be augmented by the cheapest possible set of edges AUG so that a single edge deletion does not disconnect G_0. The new approach is based on a preliminary reduction of the problem and a genetic algorithm (GA) using a binary vector to represent a set of augmenting edges and therefore a candidate solution. Two strategies are proposed to deal with infeasible solutions that do not lead to edge-biconnectivity. In the first, more traditional variant, infeasible solutions are detected and simply discarded. The second method is a hybrid approach that uses an effective heuristic to repair infeasible solutions by adding usually cheap edges to AUG until the graph augmented with AUG becomes edge-biconnected. The two GA-variants are empirically compared to each other and to another iterative heuristic for the E2AUG problem using instances involving up to 1270 edges.

1 Introduction

When designing communication networks, a minimum spanning tree is usually the cheapest network that will allow a given set of sites to communicate. However, such a network is not robust against failures, since it might not survive the break of even a single link or site. For many communication structures, an important issue besides the minimization of connection costs is reliability. The network should be robust against failures in connections or switching nodes in the sense that any two nodes do not loose connection in case of up to a certain maximum number of simultaneous failures. To accomplish this, redundant communication routes must exist for any pair of nodes.

In graph theory, the terms *vertex-connectivity* and *edge-connectivity* are used to describe this kind of robustness. A connected, undirected graph $G(V, E)$ has edge-connectivity $C_E(G)$ ($C_E(G) \geq 1$) if at least $C_E(G)$ edges need to be deleted

in order to separate G into disconnected components. Similarly, the graph has vertex-connectivity $C_V(G)$ ($C_V(G) \geq 1$) if at least $C_V(G)$ vertices with their adjacent edges must be deleted for disconnecting G. Note that C_V is always less than or equal to $C_E(G)$, since at most one incident vertex for any of $C_E(G)$ edges disconnecting G need to be deleted [2]. Furthermore, $C_E(G)$ is always less than or equal to the minimum-degree of all vertices V [20].

In this article, we concentrate on the *edge-biconnectivity augmentation* (E2AUG) problem, which is stated as follows. Given are a weighted, undirected graph $G(V, E)$ with edge-connectivity $C_E(G) \geq 2$ and a spanning subgraph $G_0(V, E_0)$, $E_0 \subset E$ with $C_E(G_0) = 1$. Each edge $e \in E$ has an associated weight $w(e) > 0$. The goal is to identify a set of augmenting edges $AUG \subset E \setminus E_0$ with minimum total weight

$$W(AUG) = \sum_{e \in AUG} w(e) \tag{1}$$

such that graph $G_{AUG}(V, E_0 \cup AUG)$ is edge-biconnected, i.e. $C_E(G_{AUG}) \geq 2$. In G_0, an edge $e \in E_0$ is called a *bridge* if its deletion disconnects G_0. G_{AUG} may therefore have no bridges. Note that the E2AUG problem is also called *bridge-connectivity augmentation problem* [4].

Besides the design of communication networks, this problem is also important to VLSI floorplanning [19]: An electronic circuit can be interpreted as a graph whose vertices are the (rectangular) functional units and whose edges are the interconnections between the units. If the graph has a so-called *rectangular dual* (a planar embedding of a dual of the graph such that each face and the total graph enclosure are rectangles), the most area-efficient chip-layout of the units can be determined efficiently. It is known that a rectangular dual exists if the graph is maximal planar and does not have complex triangles [19]. In general, a graph representing an electronic circuit will not fulfill these requirements; the graph must be augmented by additional vertices and edges. The most complex part of this augmentation problem can be transformed into the E2AUG problem.

Eswaran and Tarjan [3] proposed a polynomial-time algorithm for the special case when the weights $w(e), e \in E$, are all equal and G is a complete graph. However, for the general case with different weights, they showed that the problem is $\mathcal{NP}$-complete, see also [5, 8]. The problem even remains $\mathcal{NP}$-complete if weights are chosen from set {1,2} only [4]. In general, it is computationally too expensive to solve larger problem instances to optimality using exact techniques like branch-and-bound. Therefore, heuristics which are able to find high-quality suboptimal solutions in polynomial time are of interest.

The next section gives an overview of previous work related to the E2AUG problem. A new genetic algorithm approach is proposed in Sect. 3 and hybridized by including a problem-specific repair and improvement heuristic in Sect. 4. Empirical results for both variants are presented in Sect. 5. Especially the new hybrid approach finds good solutions with high confidence that are usually better or as good as those of another recently proposed iterative heuristic. Finally, conclusions are drawn in Sect. 6.

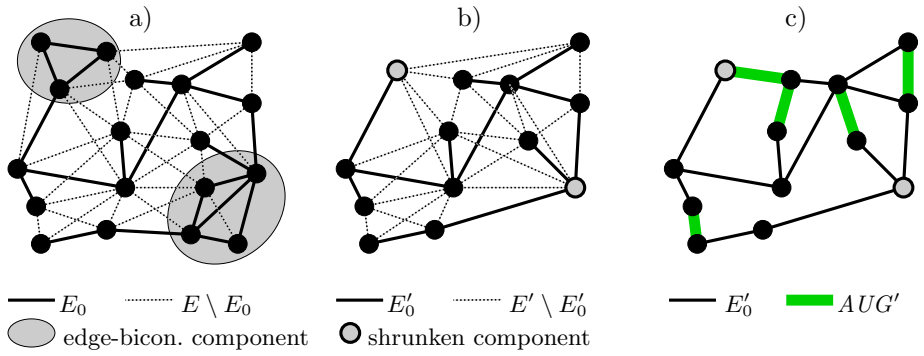

Fig. 1. An example for the problem reduction and augmentation: (a) given graphs $G(V, E)$ and $G_0(V, E_0)$, (b) shrinking of existing edge-biconnected components into single vertices, and (c) a feasible solution (AUG')

2 Related Work

Frederickson and Jájá [4] proposed an approximative algorithm for the E2AUG problem which is based on the following steps:

Firstly, the problem is simplified by detecting all already edge-biconnected components in G_0 and shrinking their vertex sets in G and G_0 into single new vertices, see Fig. 1. Edges that connect vertices of the same component can be discarded. Furthermore, among the edges that connect the same pair of components only the minimum weight edge must be retained, i.e. in case of G_0 the bridges. Let $G'(V', E')$ and $G'_0(V', E'_0)$ be these reduced versions of $G(V, E)$ and $G_0(V, E_0)$, respectively. In this way, G'_0 will always be a spanning tree and many edges from $E \setminus E_0$ can usually be discarded from further consideration as augmenting edges. Note that for each edge in E', a reference to the corresponding edge in E is stored for being able to efficiently transform a final augmenting edge set AUG' for the reduced graphs to the corresponding set AUG for the original graphs.

In the next step, G'_0 is interpreted as a directed tree in such a way that every node has a path to a selected root node r. Then, a minimum weight branching from node r, i.e. a directed minimum spanning tree including paths from r to all other nodes, is determined. Using this minimum weight branching, a set AUG' (AUG) of edges augmenting G'_0 (G_0) to become edge-biconnected can be derived.

In [4], it is also shown that the total weight of the edges added by this technique is no more than twice the weight of an optimal augmentation. Subsequently, this algorithm was improved by Khuller and Thurimella [9] with regards to the time-complexity.

Khuller and Vishkin [10] proposed a similar approach for the related problem of identifying a minimum weight edge-biconnected spanning subgraph when no starting graph G_0 is given. Khuller and Raghavachari [12] proposed another algorithm using similar basic ideas for the problem of identifying a (nonspanning)

subgraph with a given edge- or vertex-connectivity for a given graph G. A survey on several related problems and approximation algorithms is given by Khuller [11]. Algorithms with better worst-case approximation factors for such problems are sometimes known when the edge weights fulfill certain conditions such as the triangle inequation. Du et al. [1] presented an approach to the k-edge-connected Steiner Network problem in metric spaces.

Recently, Zhu et al. [21] proposed another algorithm for the E2AUG problem with arbitrary weights, called DROP, that is based on [4] and leads in practice to significantly better results than the previous approaches, although it has the same worst-case approximation factor of two. Instead of deriving all augmenting edges from a single minimum weight branching as in [4], an iterative process is used which fixes only one augmenting edge at a time based on some measurement of how useful a particular edge is. After fixing one edge, its weight is reduced to zero and a new minimum weight branching is derived according to this modification. This process continues until the minimum weight branching contains only zero-weight edges and a complete set of augmenting edges has been derived.

The first author presented in [15] and together with J. Kratica in [16] a genetic algorithm (GA) for the vertex-biconnectivity augmentation problem. This approach represents solutions by binary vectors. Infeasible solutions that are not vertex-biconnected are either discarded or repaired by a heuristic. We used this algorithm and the GANP framework [14] as a first basis for approaching the E2AUG problem. However, various far reaching modifications turned out to be necessary or important to obtain the efficient algorithm proposed in the next sections.

3 The Basic Genetic Algorithm

As proposed by Frederickson and Jájá [4], we perform a pre-processing which reduces the graphs G_0 to G'_0 and G to G' by shrinking all already edge-biconnected components in G_0 into single new vertices (Fig. 1). We remove all selfloops and multiple edges, and consider only the cheapest edges connecting two different edge-biconnected components. The set of edges that might act as augmenting edges is then $E' \setminus E'_0$.

The main structure of the proposed algorithm corresponds to a traditional generational GA with overlapping populations [6]. Its operators and properties are described in the following.

Encoding: A candidate solution is represented by a vector $\boldsymbol{x}$ of $l = |E' \setminus E'_0|$ binary genes $x_i \in \{0, 1\}$, $i = 1, \ldots, l$. Each gene x_i is associated with an edge from $E' \setminus E'_0$ and indicates whether the edge is included in the set of augmenting edges AUG' ($x_i = 1$) or not ($x_i = 0$).

Initialization: Initial solutions are created randomly by setting each gene x_i independently with probability 7/8 to 1. In this way, an initial augmented graph

is usually dense and the probability that the solution is feasible, i.e. the augmented graph is edge-biconnected, is high. But nevertheless, it is not guaranteed that only feasible solutions are generated.

Objective function: For each new chromosome, the set of augmenting edges AUG' is determined, and an edge-biconnectivity check is performed on the augmented graph $G'_{AUG} = (V', E'_0 \cup AUG')$. This test involves a depth-first search (DFS) on the graph and is derived from Tarjan's algorithm to check vertex-biconnectivity [18]:

During the DFS all vertices are numbered in the order they are reached. The edges which the DFS follows to get to yet unreached nodes form a directed tree (precisely a spanning arborescence with the starting node as root). For each node j, a so-called *low-value* is determined, which is the smallest node reachable from j by traversing zero or more tree edges followed by at most one other edge (*back-edge*) [18]. If a node j with a low-value greater than the number of its parent p exists, the edge (p, j) is a bridge, and the graph is therefore not edge-biconnected. Otherwise, if no such node exists, the graph is edge-biconnected and AUG' represents a feasible solution. The time-complexity of the whole test is $O(|E'_0 \cup AUG'|)$. A similar algorithm for testing edge-biconnectivity can also be found in [11].

In our basic GA, any infeasible solution is discarded. The objective value of a feasible solution is the sum of the weights of all augmenting edges $W(AUG) = W(AUG')$, see Eq. 1.

Selection and population replacement policy: Classic fitness proportional selection with different variants of scaling and tournament selection were empirically tested. The tournament selection turned out to be more robust for this application. In the experiments documented in Sect. 5, the population size was $P = 150$, and tournament selection was applied with a group size of 5.

During each generation, the worst 1/3 of the population is replaced by new solutions generated by means of crossover and mutation. In order to avoid premature convergence, each new solution is checked if it is already contained in the population and discarded in that case.

Crossover: Uniform crossover according to [17] is applied with a certain probability ($p_c = 85\%$ in the examples of Sect. 5). The value of each gene of an offspring is determined independently by inheriting it from the first parent with a probability of 30% and otherwise from the second parent. In this way, about 30% of the genes are exchanged.

Mutation: Each gene of a newly created solution is mutated with a certain probability p_m. Usually, the diversity of the genetic material is large at the beginning of a run and decreases with the time. We adapt the mutation rate during a run to promote a fast convergence to good solutions during the first generations and to introduce more diversity for escaping from local optima during later stages. The mutation probability at generation t is

$$p_m(t) = p_{m0} + (p_{m1} - p_{m0})\, 2^{-t/\gamma} \tag{2}$$

```
(1) procedure repair(V', E', E'_0, var AUG');
(2) begin
(3)   while (bridge ← FindBridge(V', E'_0 ∪ AUG')) do
(4)     {C_1, C_2} ← FindConnectedComponents(V', E'_0 ∪ AUG' \ {bridge});
(5)     for e ∈ E' \ (E'_0 ∪ AUG') sorted according to increasing w(e) do
(6)       if e connects C_1 with C_2 then
(7)         AUG' ← AUG' ∪ {e};
(8)         continue at (3);
(9) end;
```

Fig. 2. Pseudo-code for the greedy repair heuristic

with p_{m0} and p_{m1} denoting the lower and upper mutation rates for the beginning and ending, respectively. The parameter γ controls, how fast the mutation rate changes towards p_{m1}.

According to preliminary tests, we suggest to set the lower mutation rate to $p_{m0} = 1/(2l)$, where l is the length of the genetic code, and the upper mutation rate to $p_{m1} = 3/(2l)$.

Caching of solutions: During a GA run, some solutions are often generated repeatedly in subsequent generations. A classical GA evaluates each solution irrespectively of its repetition. To increase the efficiency, we use a caching technique [13, 14] which memorizes all newly generated solutions with their objective values. In case of a subsequent occurrence of a solution, the objective value can quickly be retrieved instead of performing a new evaluation, as long as the solution resides in the cache. The used caching technique applies a least-recently-used strategy with a hash-queue data structure [13].

4 The Hybrid Genetic Algorithm

Instead of discarding a newly generated solution that is infeasible, it can also be repaired by extending set AUG' with additional edges by a heuristic until the solution becomes edge-biconnected. For this purpose, the greedy heuristic shown in Fig. 2 is used. This procedure is performed in a Lamarckian way, therefore, the actual genotype is also modified according to the changes in set AUG'.

First, a bridge is determined by performing a depth-first search as already described in Sect. 3. If no bridge exists, graph G'_{AUG} is already edge-biconnected and the procedure terminates. Otherwise, the bridge is temporarily removed from AUG', which will disconnect the graph into two components C_1 and C_2. These components are identified by an additional depth-first search. Next, the cheapest edge from E' which is not yet contained in the augmented graph (including the bridge) and which connects C_1 with C_2 is searched for. This edge is then included in AUG'. In this way, the originally found bridge – and eventually also others – are bypassed. The algorithm restarts with checking the graph for edge-biconnectivity and identifying a bridge until the solution becomes feasible.

Table 1. Categories of test problem instances: ranges for the number of vertices and edges of G, length l of genetic code, and edge-weights $w(e)$

Category	$\|V\|$	$\|E\|$	l	$w(e)$
A	$[20, 40]$	$[43, 79]$	$[24, 40]$	$[1, \|V\|(\|V\|-1)/2]$
B	$[20, 70]$	$[81, 150]$	$[55, 81]$	$[1, \|V\|(\|V\|-1)/2]$
C	$[20, 100]$	$[205, 248]$	$[126, 182]$	$[1, \|V\|(\|V\|-1)/2]$
D	$[40, 100]$	$[384, 497]$	$[315, 398]$	$[1, \|V\|(\|V\|-1)/2]$
M	$[70, 90]$	$[312, 438]$	$[243, 349]$	$[10, 1000]$
N	$[100, 110]$	$[1203, 1270]$	$[1104, 1161]$	$[10, 50]$

For time-efficiency, all edges $E' \setminus E'_0$ are sorted according to increasing weights only once in a pre-processing step at the beginning of a run. In the worst-case, AUG' is the empty set, each edge of the augmented graph is a bridge, and $|E'_0|-1$ additional edges having always the largest weights must be included. The time-complexity of this heuristic is then $O(|E'||V'|)$. However, in practice only few bridges exist in most solutions generated by the GA and a single additional edge often eliminates several bridges. Therefore, the time-demand seems to be acceptable also for relatively large problem instances, see Sect. 5. In the following, we denote this hybrid version of the GA including the repair heuristic as HGA and the basic GA of Sect. 3 as BGA.

In contrast to BGA, it is for HGA not meaningful to create large augmentation sets leading to dense graphs as initial random solutions. This would only slow down the convergence to light-weight high quality solutions. Instead, initial solutions should now contain only few augmenting edges since the repair heuristic will add necessary edges of usually small weights. Therefore, each gene is now set to 1 with probability 1/16 only and to 0 otherwise.

5 Empirical Results

In this section, some typical empirical results obtained by BGA, HGA, and the iterative DROP heuristic from Zhu et al. [21] (which has been applied several times with different nodes as root), are documented. Problem instances belonging to different categories were randomly created by a test data generator already used in [21].

Table 1 shows the most important characteristics of these problem categories. The graphs G of all problem instances were created randomly in such a way that they are guaranteed to be edge-biconnected. The graphs G_0 to be augmented are always randomly generated spanning trees of G. In this way, no preliminary problem reduction by shrinking already edge-biconnected components is possible and G' and G'_0 always correspond to G and G_0, respectively. Furthermore, the length of the genetic code is always $l = |E| - |E_0| = |E| - |V| + 1$.

For BGA and HGA, all strategy parameters were set as described in the previous sections. Each GA run was terminated when no better solution had been found within the last G_{conv} generations with $G_{\text{conv}} = 5000$ for BGA and

Table 2. Results for DROP, BGA, and HGA: Average objective values of final solutions $W(AUG)$ with CPU-times t and number of generations *gen* needed; totally best observed objective values $W(AUG^*)$ (always from HGA)

Problem	DROP	BGA			HGA			
	$W(AUG)$	$W(AUG)$	t [s]	*gen*	$W(AUG)$	t [s]	*gen*	$W(AUG^*)$
A1	**686**	**686**	0.08	48	**686**	0.01	1	686
A2	**496**	**496**	0.06	33	**496**	0.01	1	496
A3	6804	**6607**	0.13	50	**6607**	0.07	8	6607
A4	1642	**1538**	0.16	53	**1538**	0.01	1	1538
A5	**4281**	**4281**	0.48	180	**4281**	0.01	1	4281
Avg(A)	*2782*	*2722*	*0.18*	*73*	*2722*	*0.02*	*2*	*2722*
B1	**15512**	**15512**	0.45	192	**15512**	0.02	1	15512
B2	**223**	**223**	1.14	485	**223**	0.08	15	223
B3	**7168**	7223	1.39	555	**7168**	0.01	1	7168
B4	**5984**	**5984**	1.59	548	**5984**	0.28	21	5984
B5	**13522**	13556	3.11	738	**13522**	0.34	15	13522
B6	19026	19113	2.58	603	**19022**	4.50	188	19022
Avg(B)	*10239*	*10268*	*1.71*	*520*	*10238*	*0.87*	*40*	*10238*
C1	**22309**	22399	4.62	1261	**22309**	1.26	31	22309
C2	**38191**	38396	7.27	1914	**38191**	1.34	24	38191
C3	59776	59271	12.46	2973	**59129**	3.40	53	59129
C4	**790**	800	8.51	2510	**790**	0.18	18	790
Avg(C)	*30266*	*30217*	*8.21*	*2165*	*30105*	*1.54*	*32*	*30105*
D1	**4932**	5228	27.94	4915	**4932**	2.22	31	4932
D2	**899**	930	22.30	4196	**899**	1.57	58	899
D3	20411	21429	53.33	8612	**20351**	7.52	83	20321
D4	**8142**	9340	47.00	7602	**8142**	6.75	80	8142
D5	19602	21849	63.14	8939	**19423**	32.22	363	19355
Avg(D)	*10797*	*11755*	*42.74*	*6853*	*10749*	*10.06*	*123*	*10730*
M1	3010	3172	41.84	7068	**2940**	14.68	308	2940
M2	4610	4791	54.54	8510	**4600**	14.85	208	4600
M3	5040	5299	48.21	6346	**4980**	3.44	33	4980
Avg(M)	*4220*	*4421*	*48.20*	*7308*	*4173*	*10.99*	*183*	*4173*
N1	398	490	219.03	15631	**393**	230.38	1907	383
N2	450	544	201.90	13600	**439**	544.94	2408	432
Avg(N)	*424*	*517*	*210.47*	*14616*	*416*	*387.66*	*2157*	*408*

$G_{\text{conv}} = 2000$ for HGA. The parameter γ, which controls the adaption of the mutation rate, was set to $l/2$ for BGA and to $l/4$ for HGA.

Table 5 shows results for the three algorithms. For BGA and HGA, the times t (in seconds on a Pentium III/500MHz PC) and generations *gen* indicate when the best-of-run solutions having objective value $W(AUG)$ had actually been encountered. All these values are average values determined from 10 independent

runs per problem instance. The best average objective values are always printed bold. In addition, also the totally best objective values from the 10 HGA-runs per instance ($W(AUG^*)$) are shown.

For the smaller problem categories A to C, the differences in the quality of final solutions between all three algorithms are only small. Especially for category A, HGA is able to identify the best solutions often already in the initial generation due to its heuristic, and the needed CPU-times are only fractions of a second. In some cases of category A, BGA and HGA are able to find better solutions than DROP. For the larger problem instances in categories C to N, BGA cannot compete with DROP and HGA, but HGA obtains in all cases solutions either equally good or better than those of DROP. Therefore, the repair heuristic included in HGA proves to be highly effective.

It is remarkable that HGA identifies good solutions with high confidence. For many problem instances, all or most of the 10 runs per problem instance found identical or equally good solutions. Note also that due to the heuristic, the number of needed generations to identify good solutions is dramatically reduced in comparison to BGA. Also regarding the CPU times, HGA is superior to BGA in most cases.

6 Conclusion and Future Work

A new GA-based approach for augmenting a given graph G_0 with an as cheap as possible set of additional edges so that the graph becomes edge-biconnected was proposed. By applying the preliminary shrinking of already edge-biconnected components into single new vertices, the set of edges that must be considered for augmentation can often greatly be reduced. Within the GA, candidate solutions are represented by binary vectors, and infeasible solutions are either discarded (in the basic GA) or repaired by a heuristic that adds cheap edges for detected bridges until the solution becomes edge-biconnected (HGA).

In the presented empirical study involving underlying graphs with up to 110 nodes and 1270 edges, the hybrid approach proved to give significantly better results than the basic GA and a recently proposed iterative heuristic [21]. Although quality differences between the final solution of the hybrid GA and the heuristic from [21] are only small, it is remarkable that HGA's solutions are nearly always better or equally good.

Future work should include the adaption of the algorithm to other graph-augmentation problems such as the more general k-edge-connectivity augmentation and the k-vertex-connectivity augmentation problems.

Acknowledgements

We thank Samir Khuller for giving us important informations related to the edge-connectivity augmentation problem and An Zhu for providing us the problem-instance generator as well as her results obtained by the DROP-algorithm ([21]).

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

References

1. Du X., Hu X., Wong C. K.: On Shortest k-Edge-Connected Steiner Networks in Metric Spaces, Journal of Combinatorial Optimization **4(1)**, (2000), 99–107
2. Diestel R.: Graph Theory, 2nd edition, Graduate Texts in Mathematics 173, Springer, New York, (2000)
3. Eswaran K. P., Tarjan R. E.: Augmentation Problems, SIAM Journal on Computing **5**, (1979), 653–665
4. Frederickson G. N., Jájá J.: Approximation algorithms for several graph augmentation problems, SIAM Journal on Computing **10(2)**, (1981), 270–283
5. Garey M. R., Johnson D. S.: Computers and Intractability: A Guide to the Theory of $\mathcal{NP}$ Completeness, Freeman, (1979)
6. Goldberg D. E.: Genetic Algorithms in Search, Optimization and Machine Learning, Addison-Wesley, MA, (1989)
7. Jungnickel D.: Graphs, Networks and Algorithms, Springer, Berlin, (1999)
8. Kahn V., Crescenzi P.: A compendium of $\mathcal{NP}$ optimization problems, `www.nada.kth.se/theory/problemlist.html`, (1999)
9. Khuller S., Thurimella R.: Approximation Algorithms for Graph Augmentation, Journal of Algorithms **14(2)**, (1993), 214–225
10. Khuller S., Vishkin U.: Biconnectivity Approximations and Graph Carvings, Journal of the ACM **41(2)**, (1994) 214–235
11. Khuller S.: Approximation Algorithms for Finding Highly Connected Subgraphs, in Approximation Algorithms for $\mathcal{NP}$-hard problems, ed. Hochbaum D., PWS Publishing Company, (1996)
12. Khuller S., Raghavachari B.: Improved Approximation Algorithms for Uniform Connectivity Problem, Journal of Algorithms **21(2)**, (1996), 434–450
13. Kratica J.: Improving Performances of the Genetic Algorithm by Caching, Computers and Artificial Intelligence **18(3)**, (1999), 271–283
14. Kratica J.: Parallelization of Genetic Algorithms for Solving Some $\mathcal{NP}$-complete Problems, PhD thesis (in Serbian), Faculty of Mathematics, Belgrade, (2000)
15. Ljubic I.: Application of Genetic Algorithms in Graph Connectivity Problems, MS thesis (in Serbian), Faculty of Mathematics, Belgrade, (2000)
16. Ljubic I., Kratica J.: A Genetic Algorithm for the Biconnectivity Augmentation Problem, submitted to the IEEE Congress on Evolutionary Computation, San Diego, CA, (2000)
17. Syswerda G.: Uniform Crossover in Genetic Algorithms, Proc. of the 3rd Int. Conference on Genetic Algorithms, Morgan Kaufmann, San Mateo, Calif., (1989), 2–9
18. Tarjan R. E.: Depth First Search and Linear Graph Algorithms, SIAM Journal of Computing **1**, (1972), 146–160
19. Tsukiyama S., Kioke K., Shirakawa I.: An Algorithm to Eliminate All Complex Triangles in a Maximal Planar Graph for Use in VLSI Floorplanning, in Algorithmic Aspects of VLSI Layout, ed. Sarrafzadeh M., Lee D. T., World Scientific Publishing, (1993)
20. Veljan D.: Combinatorics and Graph Theory, Školska knjiga, Zagreb, (1989)
21. Zhu A., Khuller S., Raghavachari B.: A Uniform Framework for Approximating Weighted Connectivity Problems, Proc. of the 10th Annual ACM-SIAM Symposium on Discrete Algorithms, Baltimore, (1999), 937–938