A Genetic Algorithm for the Biconnectivity Augmentation Problem

Ivana D. Ljubic
Institute of Computer Graphics
Vienna University of Technology
Karlsplatz 13/1861
1040 Vienna, Austria
ljubic@apm.tuwien.ac.at

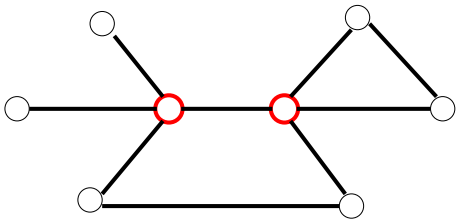
Jozef J. Kratica
Mathematical Institute
Serbian Academy of Sciences
Kneza Mihajla 35/I
11000 Belgrade, Serbia
jkratica@matf.bg.ac.yu

July 2000
IEEE Congress on Evolutionary Computation

Overview

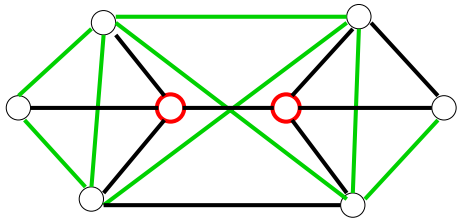
1. The Vertex-Bicconectivity Augmentation Problem
2. Straight-forward GA approach with caching
3. Hybridization with a greedy problem-dependent heuristic
4. Experimental comparisons
5. Conclusions

1. The Vertex Bicconectivity Augmentation Problem

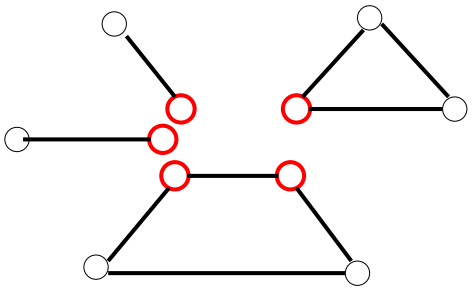


Currently existing network

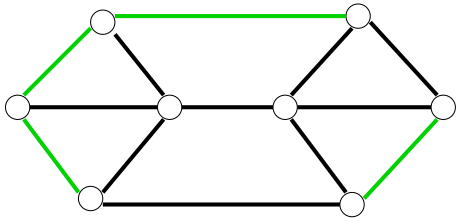
Cut points



Feasible edges



Biconnected components - blocks



Possible biconnectivity augmentation

$\rightarrow$ Applications: VLSI design, World Wide Web, telecomunication networks...

Matematical Definition

- Given: undirected, 2(node)-connected weighted graph $G(V, E)$, and $E_0 \subset E$ set of fixed edges, $w : E \mapsto R^+$.
- Goal: Find the set of augmenting edges $AUG \subset E \setminus E_0$ such that
  1. Graph $G_{AUG}(V, E_0 \cup AUG)$ is node-biconnected, and
  2. The set AUG is of minimal weight, i.e.

$$\textit{minimize} \quad \text{obj}(G_{AUG}) = \sum_{e \in AUG} w(e).$$

$\rightarrow$ The problem is NP-hard (Eswaran, Tarjan, 1972).
$\rightarrow$ The problem stays NP-hard even when $G(V, E_0)$ is connected (Frederickson, Jájá, 1981).

2. Straight-forward GA approach with caching

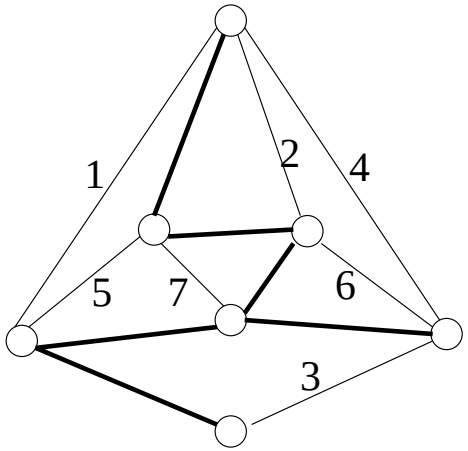
- Apply a **direct encoding:** $\vec{x} = (x_1, x_2, \ldots, x_l)^T$, $x_i \in \{0, 1\}$, $(i = 1, \ldots, l)$, such that:

$$e_i \in AUG \Leftrightarrow x_i = 1.$$

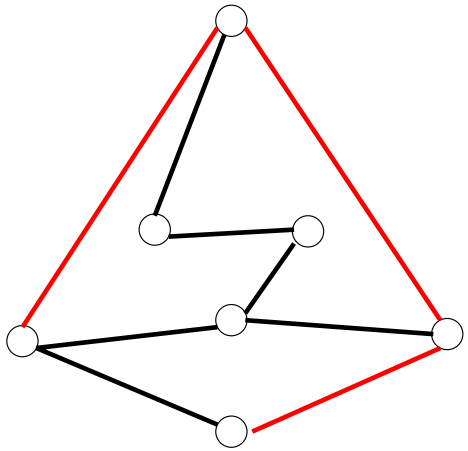
- Every individual represents a graph - potential solution of the problem.

- GANP implementation (Kratica, 2000).

Example of encoding

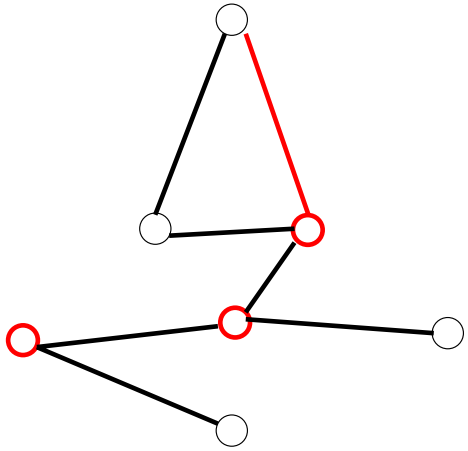


Original graph



Order the edges: 1 2 3 4 5 6 7 8

Genetic code: 1 0 1 1 0 0 0 0



Genetic code: 0 1 0 0 0 0 0

Fixed edges

Feasible edges

Augmented edges

Cut points

Main GA properties

- **Objective value:** Tarjan's (1972) algorithm for checking biconnectivity is applied. Time complexity $O(|E_0 \cup AUG|)$.
- **Discarding infeasible solutions:** Once, when we determine that an individual is not biconnected, we eliminate it from the mating process.
- **Caching:**
  $\rightarrow$ No influence on the GA's results - only to improve the run-time performance.
  $\rightarrow$ Avoid the GA's attemption to compute the same objective values repeatedly. Instead of it, remember and reuse objective values.

- **Selection:** Tournament selection (tournament size: 3,5,7).
- **Crossover:** Uniform crossover.
- **Mutation:** Changed during the GA execution, according to the formula:

$$p_m = p_s + (p_e - p_s)2^{-\frac{i}{c}}.$$

- **Initialization** Each gene is randomly initialized with setting each bit to 0 with probability 1/8.

$\rightarrow$ Main weakness of this approach: too many infeasible solutions could lead to slow or premature convergence.
$\rightarrow$ Penalization or repair of infeasible solutions is necessary.

3. Hybridization with a greedy problem-dependent heuristic

- Each generated infeasible solution is repaired by a greedy heuristic.
- Based on:
  1. Tarjan's (1972) algorithm for checking biconnectivity of connected graphs (in $O(|E|)$ time), and
  2. Kruskal's (1956) greedy algorithm for creating minimum spanning trees (in $O(|E| \log |E|)$ time).

- **Preprocessing:** Sort all edges in $E \setminus E_0$ according to increasing weight, i. e. $E \setminus E_0 = \{e_1, e_2, \ldots, e_l\}$, with

$$i < j \Leftrightarrow w(e_i) \leq w(e_j), \quad l = |E \setminus E_0|.$$

A greedy, problem-dependent heuristic

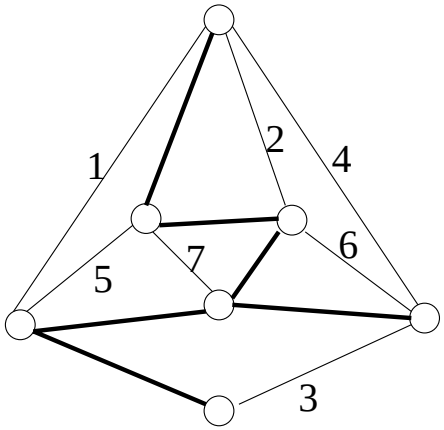
- $AUG' = \emptyset$; $E'_0 = E_0 \cup AUG$,
- While there are cut points in $G' = G(V, E'_0)$:
  - Randomly choose a cut point a and remove it from G'.
  - Make shrinked graph G_s: shrink connected components of $G' - \{a\}$, and consider only the minimum weight edges connecting two different components.
  - Apply Kruskal's Minimum Spanning Tree algorithm on G_s to find AUG_s.
  - Determine original edges from AUG_s and add them to AUG'.

$\rightarrow$ The worst-time complexity: $O(|V||E| \log |V|)$.

$\rightarrow$ Max number of iterations is the number of cut points.

$\rightarrow$ In practice, algorithm performs faster (cut points are often eliminated simultaneously).

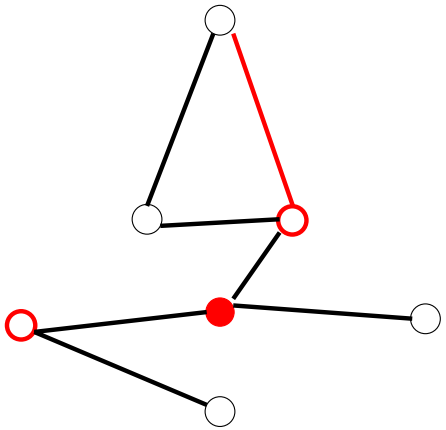
One loop inside the heuristic



Original graph

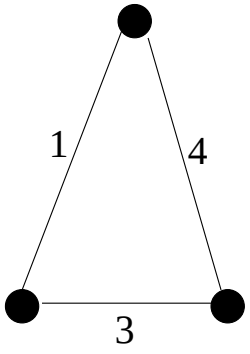
Fixed edges

Feasible edges



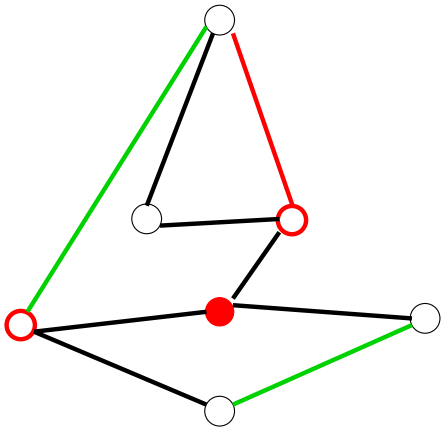
Graph G’

Remove it from G’



After shrinking

Blocks



Graph G’+AUG’

AUG’

4. Experimental Comparisons

Population size	- 150
Generation replacement policy	- overlapping populations with elitist strategy, - only 1/3 of population is generated anew, - the worst solutions are replaced, - duplicate phenotypes are discarded - fitness of the elitist individual is temporarily decreased by the average fitness of the population
Recombination	uniform crossover with $p_{unif} = 0.3$ and $p_{cross} = 0.85$
Mutation	starting value: $\begin{cases} p_s = 1/2l, & \text{if } 1/2l \geq 0.01 \\ 0.01, & \text{otherwise} \end{cases}$ end value $p_e = 0.01$

Termination	best known solution obtained, or maximal number of generations reached
Number of generations	from 100 - 30000 (in PGA case) from 50 - 2000 (in HGA case)

The test examples

$\rightarrow$ All initial graphs are the spanning trees.

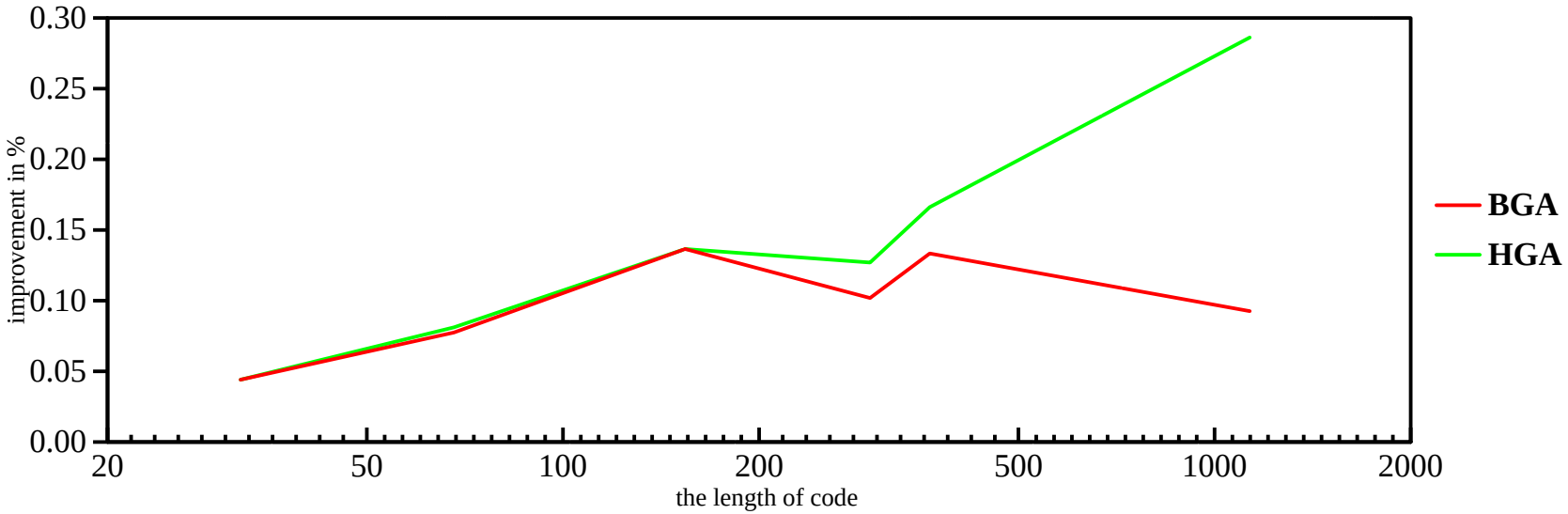
$\rightarrow$ Graphs are made by Zhu's (1999) generator.

Category	size	n	m	l	$w(e)$
A	5	$[20, 40]$	$[43, 79]$	$[24, 40]$	$[1, n(n-1)/2]$
B	5	$[20, 70]$	$[81, 150]$	$[55, 81]$	$[1, n(n-1)/2]$
C	4	$[20, 100]$	$[205, 248]$	$[126, 182]$	$[1, n(n-1)/2]$
D	5	$[40, 100]$	$[384, 497]$	$[315, 398]$	$[1, n(n-1)/2]$
M	3	$[70, 90]$	$[312, 438]$	$[243, 349]$	$[10, 1000]$
N	2	$[100, 110]$	$[1203, 1270]$	$[1104, 1161]$	$[10, 50]$

(properties of the problem instances)

Category	Greedy	BGA			HGA			Impr.
	$w(AUG)$	$w(AUG)$	$t[s]$	gen	$w(AUG)$	$t[s]$	gen	%
A	2994	2862	0.7	25	2862	0.5	8	6.4
B	9270	8552	3.8	107	8518	2.6	22	7.6
C	35049	30263	16.4	970	30263	35.6	128	13.0
D	12873	11156	556.3	11977	10734	549.3	513	14.5
M	4780	4293	614.3	10639	4173	67.7	128	13.0
N	594	539	1520.62	21221	424	1312.69	1176	28.2

(the best values from 10 consecutive executions of GA's)



Improvements of Greedy algorithm

5. Conclusions

- Both BGA and HGA gives significantely better results then the greedy heuristic alone.
- The straight-forward approach works well for small problem instances.
- HGA is better concerning:
  - quality of the final solution,
  - needed number of generations,
  - corresponding execution times.

Future work

- The same GA approach could be easily adapted to other graph connectivity problems (Ljubic, Raidl, Kratica 2000).
- Examination of other heuristics.
- Preprocessing: shrink biconnected components of $G(V, E_0)$ and reduce the size of $|E \setminus E_0|$.
- Examination of problem-dependent GA operators.