Performance analysis of the partial use of a local optimization operator on the genetic algorithm for the Travelling Salesman Problem

Milan Djordjevic, Marko Grgurovič and Andrej Brodnik
Department of Information Science and Technology, University of Primorska, Koper, Slovenia

Abstract

Background: The Travelling Salesman Problem is an NP-hard problem in combinatorial optimization with a number of practical implications. There are many heuristic algorithms and exact methods for solving the problem. **Objectives:** In this paper we study the influence of hybridization of a genetic algorithm with a local optimizer on solving instances of the Travelling Salesman Problem. **Methods/ Approach:** Our algorithm uses hybridization that occurs at various percentages of generations of a genetic algorithm. Moreover, we have also studied at which generations to apply the hybridization and hence applied it at random generations, at the initial generations, and at the last ones. **Results:** We tested our algorithm on instances with sizes ranging from 76 to 439 cities. On the one hand, the less frequent application of hybridization decreased the average running time of the algorithm from 14.62 sec to 2.78 sec at 100% and 10% hybridization respectively, while on the other hand, the quality of the solution on average deteriorated only from 0.21% till 1.40% worse than the optimal solution. **Conclusions:** In the paper we have shown that even a small hybridization substantially improves the quality of the result. Moreover, the hybridization in fact does not deteriorate the running time too much. Finally, our experiments show that the best results are obtained when hybridization occurs in the last generations of the genetic algorithm.

Keywords: genetic algorithm, travelling salesman problem, hybridization, optimization, grafted genetic algorithm
JEL classification: C61
Paper type: Research article

Recieved: 28, April, 2012
Revised: 30, May, 2012
Accepted: 29, June, 2012

Citation: Djorjevic, M., Grgurović, M., Brodnik, A. (2012). "Performance analysis of the partial use of a local optimization operator on the genetic algorithm for the Travelling Salesman Problem", Business Systems Research, Vol. 3, No. 1, pp. 14-22.
DOI: 10.2478/v10305-012-0002-4

Introduction

Genetic Algorithms (GA) use mechanisms inspired by biological evolution (Holland, 1975; Haupt and Haupt, 2004). They are applied on a finite set of individuals called population. Each individual in a population represents one of feasible solutions in a search space. Mapping between genetic codes and the search space is called encoding and can be binary or over some alphabet of higher cardinality. Good choice of encoding is a prime condition for successful application of a genetic algorithm. Each individual in the population is assigned a value called fitness. Fitness represents a relative indicator of quality of an individual compared to other individuals in the population. Selection operator chooses individuals from the current population and takes the ones that are transferred to the next generation. Thereby, individuals with better fitness are more likely to survive in the next generation. The recombination operator combines parts of genetic code of the individuals (parents) into codes of new individuals (offspring).

The components (operators) of the genetic algorithm software system are: Genotype, Fitness function, Recombinator, Selector, Mater, Replacer, Terminator, and in our system (Local) Optimizer which is a new extended component.

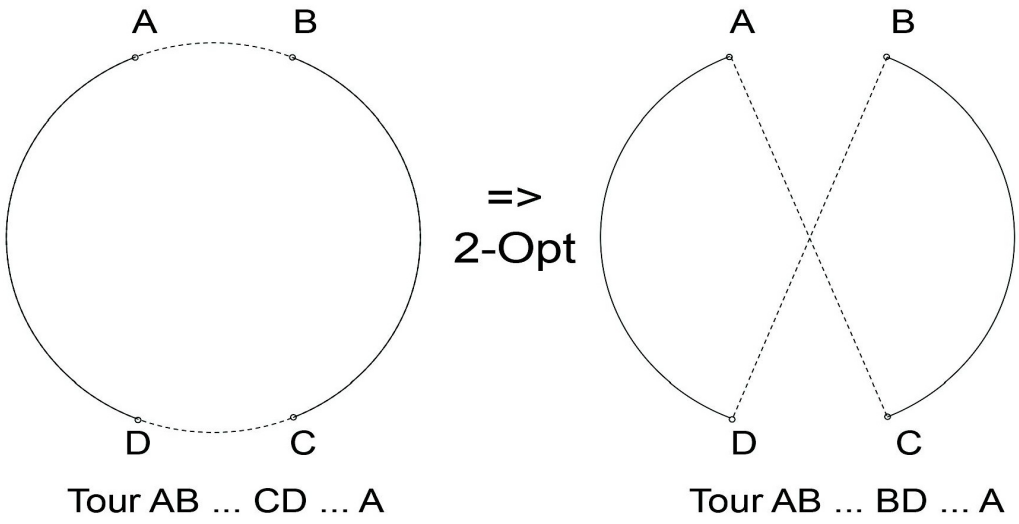
In this paper we study a well-defined problem of a Travelling Salesman Problem (TSP). The TSP is a combinatorial optimization problem on a set of cities $\{c_1, c_2, ..., c_n\}$ where for each pair $\{c_i, c_j\}$ of distinct cities a distance $d(c_i, c_j)$ is given. The goal is to find an ordering π of the cities that minimizes the quantity

$$\sum_{i=1}^{N-1} d\left(c_{\pi(i)}, c_{\pi(i+1)}\right) + d\left(c_{\pi(n)}, c_{\pi(1)}\right). \quad (1)$$

This quantity is referred to as the minimum tour length since it is the length of the tour a salesman would make to visit the cities in the order specified by the permutation π, returning at the end to the initial city. In this paper we will concentrate on a symmetric TSP in which the distances satisfy property $d(c_i, c_j)=d(c_j, c_i)$ for $1 \le i, j \le n$. More precisely, in tests we will consider the Euclidean distance version of TSP. In Euclidean TSP we have a complete graph on n cities that are defined by coordinates and the distance between cities is defined as an Euclidean distance.

TSP is frequently met in areas like logistic, scheduling, route planning, sequencing, and cellular manufacturing (Gutin and Punnen, 2002). Consequently, there is a high need for solution of practical instances of TSP. However, since the TSP is known to be NP-hard (Garey and Johnson, 1979) even under substantial restrictions, we are restricted to compute approximate solutions. There were proposed various greedy heuristics for solving TSP including nearest neighbor (Hoos and Stützle, 2005) and 2-opt. We will use the later in our algorithm as a hybridization component of a canonical GA. The main idea behind the 2-opt heuristic is to take a route that crosses itself and reorder it so that it does not cross itself any more and simultaneously reduce the tour length. An exchange step consists of removing two edges from the current tour and reconnecting the resulting two paths in the best possible way, as shown in Figure 1.

Figure 1
Elimination of crossing in 2-opt heuristic.



Source: Author's illustration

Although the 2-opt algorithm (Helsgaun, 2000; Engels and Manthey, 2009) performs well and can be applied to TSP with many cities, it finds only a local minimum. Furthermore, for the general (i.e. non-Euclidean) version of TSP it is known that there is no upper bound on a quality of a heuristic algorithms unless P=NP (Garey and Johnson, 1979). Nevertheless, we will be interested in relative quality of our algorithm on a given set of samples.

In practice one of the most frequently used programs for solving TSP is Concorde (Applegate, Bixby, Chvátal and Cook, 2001). It is based on the branch and cut method (Hahsler and Hornik, 2007). Besides it there is a range of solutions based on meta-heuristics including life-inspired approaches like ant colonies (Dorigo and Gambardella, 1997) and GA (Freisleben and Merz, 1996). Our algorithm is based on the later.

Although there are known approaches that tried to combine canonical GA with local optimization (cf. Memetic Algorithms in (Merz and Freisleben, 2001), or hybridization of GA in (Sels and Vanhoucke, 2011)), our solution is novel in two respects. First, we include local optimization as one of the operators of GA and secondly we do not use it in all generations.

The rest of the paper is organized as follows. First we introduce the general framework of our algorithm including the principle of methodology and present the procedure of hybridization of TSP solving genetic algorithm. The third section describes two experiments and instances of TSP used in the experiments. The section is followed by experimental results, their analysis and discussion. The paper concludes by summarizing the results, conclusion and suggestions for further research.

Methodology

Grafting in botany is when the tissues of one plant are affixed to the tissues of another. Grafting can reduce the time to flowering; shorten the breeding program, etc. Similarly we introduced into a canonical GA a local optimizer – we grafted GA or we hybridized it. This way we (locally) optimize each genome in an evolution process. There exist a number of local optimizers, which can be used on their own as a greedy solution to NP-hard problems – e.g. (Freisleben and Merz, 1996) used a k-opt heuristics. There exists even its hardware implementation (Hoos and Stützle, 2005).

The pseudo-code of our Grafted Genetic Algorithm (GGA) is:

1. t=0
2. p(t):=Initialize()
3. q(t):= Evaluate(P(t))
4. **While** (q(t) < $q_{expected}$) **and** (t < t_{max})
5. sel:= Select(P(t))
6. mat:= Mate(sel)
7. rec:= **for each** pair m Є mat **do** Recombine(m)
8. loc:= **for each** genome r Є rec **do** Optimize(r)
9. P(t+1):= Replace(loc, P(t))
10. q(t+1):= Evaluate(P(t+1))
11. t:=t+1
12. **EndWhile**

As usual, our algorithm stops either when the expected quality of solution is reached or when the maximum number of generations t_{max} is passed. The former can be measured in various terms starting from the absolute quality value to the relative diversification of population (e.g. standard deviation). On the other hand the later is measured either in the number of generations or in the total time elapsed.

We studied two versions of recombination (line 7 in the algorithm). The first, *Edge map crossover, emx* for short (Merz and Freisleben, 2001), uses a so-called edge map *EM[*]* that contains a list of neighbouring cities for each city. First the operator for each edge *(u, v)* in a parent genomes *A* and *B*, adds v to the edge-map list *EM[u]* and u to *EM[v]* for a symmetric version of TSP. Then the operator works as follows:

1. Pick a random city to be the current location *u*.
2. Remove the current location u from all edge map lists *EM[*]*.
3. If the current location *EM[u]* still has remaining edges, go to step 4, otherwise go to step 5.
4. Choose the new current location u' from the edge map list *EM[u]* as the one with the shortest edge map list *EM[u']*. *Set u:= u'* and go to step 2.
5. If there are left any locations, choose as the new current location u' the one with the shortest edge map list *EM[u']*. *Set u:= u'* and go to step 2.

The second recombination operator we studied was a *distance preserving crossover, dpx* for short (Freisleben and Merz, 1996). It creates a new tour (offspring) preserving the same distance in the number of edges to both parents. In detail, the operator dpx creates an offspring *C* from parents *A* and *B* as follows:

1. Pick at random parent (wlog. *A*) and copy it to *C*.
2. If *(u, v) Є C* and *(u, v) € B* delete it from *C*. At this point *C* contains fragments of connected cities *($u1 \rightarrow v_1$), …, ($u_k \rightarrow v_k$)*, where in some fragments *ui = vi.* We call set of ui and vi the end-points of *C*, EP_C.
3. Pick at random a city x from EP_C and delete it from EP_C.

4. Pick from EP_C the closest city y to x so that there is edge *(x, y)* neither in *A* nor in *B*. Delete *y* from EP_C.
5. Merge fragments $(u_i \rightarrow y)$ and $(x \rightarrow v_j)$ *into* $(u_i \rightarrow v_j)$.
6. If EP_C is not empty, go to step 3.

Although both recombination operators produced offsprings from valid parents, they produced them in a random way. Because of randomization, the selection and mating (lines 5 and 6 respectively) lost their role. Moreover, the randomization is a very good source of diversification. However, we are missing in our meta-heuristic the process of specialization.

This was the reason to extend our algorithm with a specialization step – we grafted a canonical genetic algorithm with a local optimizer and obtained a *grafted genetic algorithm*, GGA (Djordjevic and Brodnik, 2011; Djordjevic, Tuba and Djordjevic, 2009). We did not use a k-opt heuristics due to its complexity, but rather its simpler version 2-opt explained in the previous section (see Figure 1). The hybridization occurs in line 8 of the pseudocode of our algorithm given above.

Experiment

For testing our strategy and comparing it to other solutions we used the instances of Euclidean distance based symmetric travelling salesman problem found in TSPLIB (Reinelt, 1991). We used relatively small instances, for which best solutions are known. The goal of this research was not to find a better algorithm, but rather to study on a controlled environment the impact of grafting of genetic algorithm.

In the first experiment we used 20 instances, with different sizes in a range from 14 to 150 cities per instance (look in Table 1). We studied our method (GGA) using two different recombination operators: an edge map crossover (GGAemx) and a distance preserving crossover (GGAdpx). As the upper and lower limits on the quality of solution we used nearest neighbour greedy heuristic and *Concorde* respectively. For the sake of completeness we compared our method also with 2-opt heuristic itself and with a canonical genetic algorithm. The main difference between our method and canonical genetic algorithm is that we use local optimizer in every generation of the algorithm.

In the second experiment we studied what happens if we do not use local optimization in all generations — in test we used it only in 10, 20, 30, 40, 50, 60, 70, 80 and 90 percents of generations. Furthermore, for each percentage we applied local optimization in three different ways: at random generations, at the initial generations and at the ending ones.

All experiments were conducted on a computer with Pentium® 2.8 GHz CPU and Windows 7 operating system. In our results we cannot directly compare the running times of different solutions as they were implemented in different programming languages. On one hand we used as a development environment for GGA the Java written EA Visualizer (Bosman and Thierens, 1999), while *Concorde* is an *AnsiC* application. However, we can compare running times of GGA for different instances and cases explained above.

Table 1
Results of five techniques for solving Euclidean TSP

Name	Greedy	2-opt	GAemx			GAdpx			GGAemx			GGAdpx			Concorde	
	quality	quality	quality	gen.	time	quality	gen.	time	qual.	gen.	time	qual.	gen.	time	opt	time
burma14	8,32%	5,71%	0%	74	3,4	0%	81	3,5	0%	7	0,6	0%	6	0,5	3323	0,1
ulysses16	10,42%	7,15%	0%	136	4,1	0%	125	4,4	0%	9	0,7	0%	9	0,7	6859	0,2
ulysses22	12,54%	7,87%	0%	1267	14,7	0%	1328	16,4	0%	8	0,6	0%	8	0,7	7013	0,2
bayg29	13,37%	6,38%	0%	1345	19,4	0%	1137	17,6	0%	13	1,3	0%	14	1,4	1610	0,3
bays29	12,87%	5,37%	0%	2185	29,2	0%	2643	34,1	0%	12	1,2	0%	12	1,2	2020	0,3
dantzig42	14,06%	7,11%	0%	4704	79,8	0%	4232	74,6	0%	10	1,3	0%	9	1,3	699	0,5
att48	13,98%	8,47%	0%	4807	85,2	0%	5213	91,3	0%	22	2,2	0%	23	2,3	33522	0,6
eil51	15,24%	7,67%	4,21%	5482	100.0+	5,23%	5489	100.0+	0%	33	3,9	0%	30	3,8	426	0,3
berlin52	14,82%	7,45%	0%	2037	33,7	4,92%	5021	100.0+	0%	15	1,7	0%	15	1,7	7542	0,4
st70	13,17%	7,84%	5,12%	5259	100.0+	5,72%	5198	100.0+	0%	20	4,1	0%	19	4,1	675	0,5
eil76	14,47%	8,15%	6,56%	5347	100.0+	7,24%	5298	100.0+	0%	53	4,5	0,19%	49	4,4	538	1,3
pr76	13,96%	9,95%	4,18%	5218	100.0+	5,36%	5191	100.0+	0%	42	4,1	0%	43	4,2	108159	1,2
gr96	16,32%	7,14%	4,98%	5191	100.0+	5,71%	5090	100.0+	0%	73	8,4	0,13%	73	8,4	55209	1,6
rat99	14,79%	7,41%	5,31%	5114	100.0+	7,12%	5011	100.0+	0%	74	11,9	0,17%	70	11,7	1211	1,7
kroA100	12,37%	8,07%	5,12%	5072	100.0+	6,58%	4971	100.0+	0%	24	3,6	0,18%	22	3,5	21282	1,7
kroB100	16,58%	7,19%	6,14%	5041	100.0+	5,92%	4816	100.0+	0%	39	5,8	0,21%	36	5,7	22141	1,7
kroC100	10,47%	11,19%	4,87%	5121	100.0+	6,78%	4923	100.0+	0,10%	34	5,3	0,19%	28	5,1	20749	1,8
kroD100	14,81%	7,74%	5,07%	4976	100.0+	8,12%	4951	100.0+	0%	31	5,6	0,29%	25	5,3	21294	1,5
lin105	16,60%	9,85%	6,72%	4756	100.0+	6,51%	4803	100.0+	0,01%	26	4,6	0,17%	25	4,6	14379	1,3
ch150	19,62%	11,72%	7,22%	4512	100.0+	8,77%	4460	100.0+	0,22%	88	15,2	0,32%	86	15,1	6528	7

Results

We present results separately for the first and for the second experiment. In both experiments we used instances of TSP from TSPLIB of various sizes. The name of the instance also contains its size (the number of cities, cf. the first columns of Table 1). Next, in the experiments we measured three quantities: the wall clock time, the number of generations and the quality of the result. The later was measured against the optimal solution obtained by *Concorde*. The quality of algorithm A is defined as

$$q_A = \frac{l_A - l_C}{l_C} \tag{2}$$

where l_C is a path length obtained by Concorde and l_A is a path length obtained by A. We express the quality always in percents, where, for example, 4% means 4% worse than *Concorde*.

Let us first look at the results of the first experiment (cf. Table 1), in which we compared our GGA against greedy algorithm and Concorde. We also compared it against canonical genetic algorithm (GA). The termination condition in all genetic algorithms was: either standard deviation of genomes was 0 (local minimum was reached) or 100 seconds time limit expired.

We first look at the quality of results, then at the running time and finally comment on a trade-off between the quality and the running time. The last column of Table 1 gives results of Concorde and actual path length in opt column. On the other side of the table is a greedy nearest neighbour approach, whose quality (column quality is computed using Equation 2) is mostly in the range between 10% and 20%. However, application of a simple 2-opt heuristic on a randomly generated tour improves the quality to approximately 10% or even better. On the other hand use of GA further improves the quality to approximately 5%. Note, that runs in cases with 70 or more cities terminated due to time limit and hence minimum was not reached. All these results were expected.

The long running time of GA was a reason to graft (or hybridize) the GA with local optimization. The result was substantial decrease in running time. In all cases for GGAemx and for GGdpx the runs were terminated upon reaching the minimum. The reached minimum was, however, the local one. Nonetheless, we showed, that the combination of two methods improved the quality of results in a synergy. The quality of result was below 1% off the optimum.

The running times in Table 1 are given for all algorithms but greedy nearest neighbour and simple 2-opt heuristics. The later ones had running time in the range between half a second and a second and a half. However, since all algorithms but Concorde were programmed in Java, their running times are not directly comparable. Nonetheless, the relative increase in time as function of a problem size can be compared, and this shows us approximately 25 times increase for GGAemx, 30 times increase for GGAdpx and even 70 times increase for *Concorde*.

Note also, that GGAemx and GGAdpx performed approximately the same, which shows that the recombination operator has no major influence on a final result.

In the second experiment we studied the influence of grafting (hybridization) on running time and quality of solution. In this experiment we used only grafted GA with edge map crossover (GGAemx) and only eleven cases from TSPLIB (cf. Table 2).

In the experiment we were increasing the number of generations in which we applied hybridization by 10 percents: from 0% — column GAemx in Table 2 till 100% — column GGAemx. Moreover, we also varied the generations in which we applied the hybridization: either at random generations (column rnd), at the beginning ones (begin) or at the ending ones (end). The column f.a. gives the running time, while the numbers in columns q give the quality computed using Equation 2.

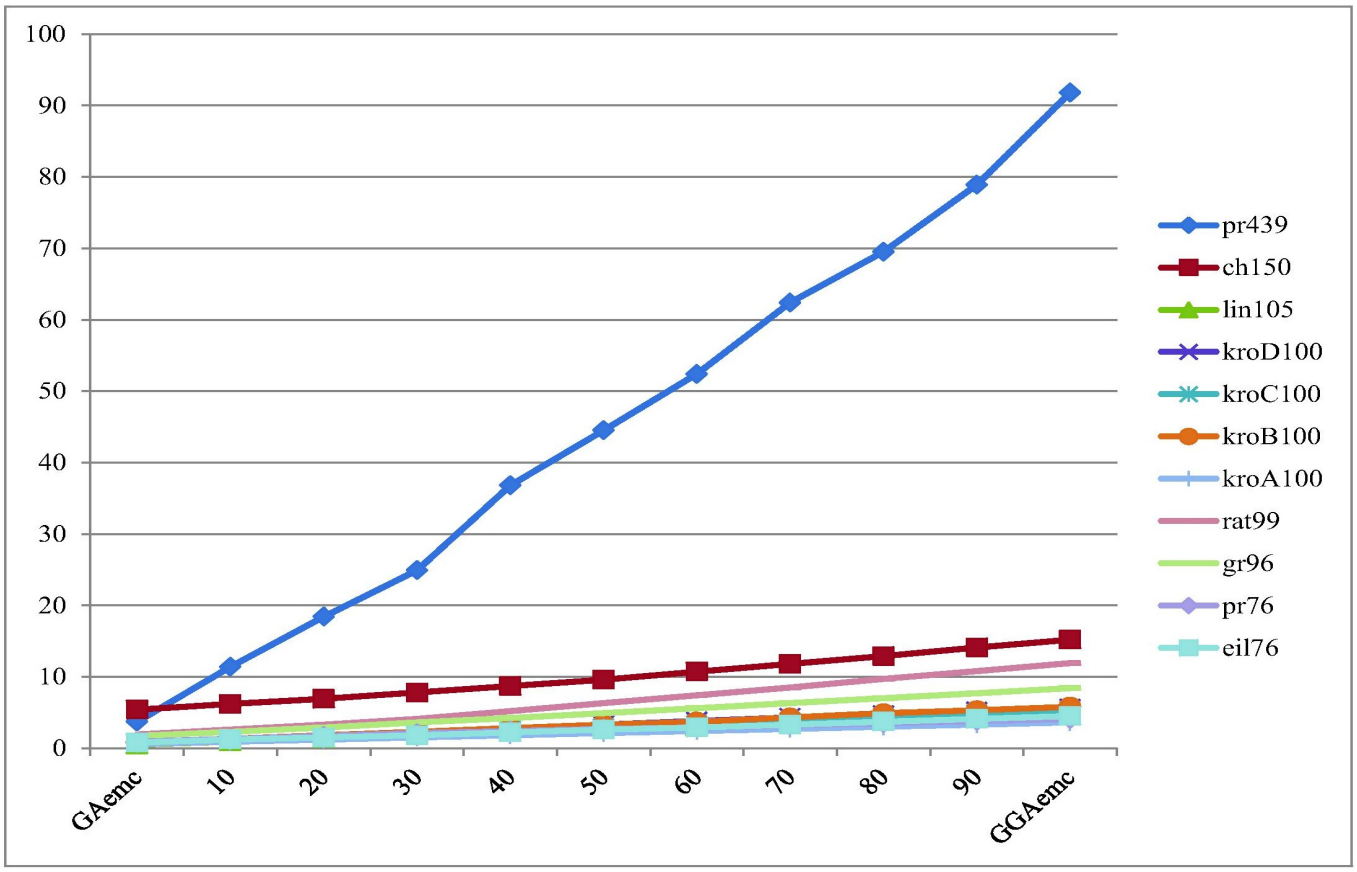
We first observe, that application of grafting in the last generations gives the best results. This is reasonable, as in general in this phase of meta-heuristics we apply mostly specialization and not that much diversification any more. On the other hand it is interesting that the worst results were obtained when grafting was applied in random generations. Nonetheless, since in practice the algorithm does not know which are the last generations, we would need to simulate the behavior. There are two possibilities how to do it: either, when time limit is reached run the algorithm for some more runs and apply hybridization or apply hybridization more and more frequently as the number of generations increases. The later approach is also in line with other meta-heuristics like simulated annealing.

The running time obviously linearly increases as we increase the amount of hybridization (see Figure 2).

Table 2
Partial grafting of a genetic algorithm.

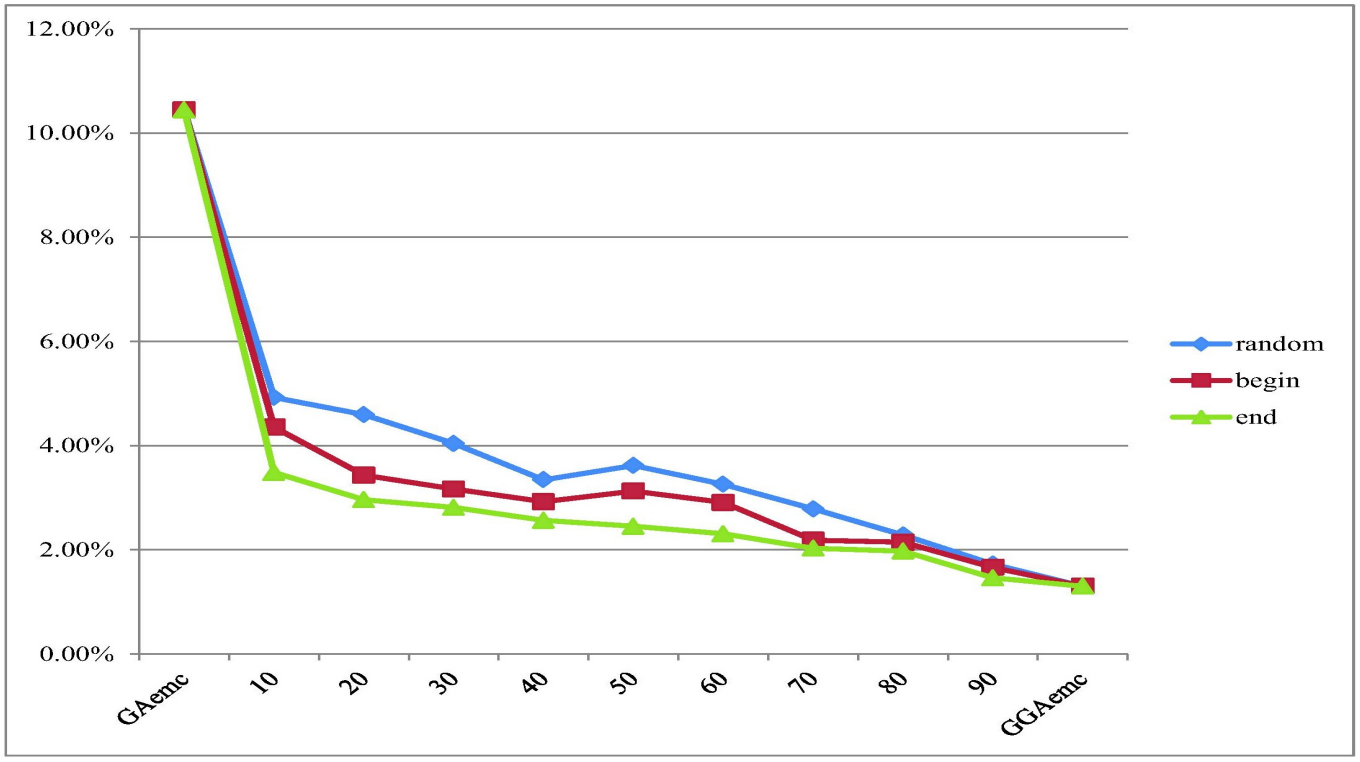
Name	GAemx		10				20				30				40				50			
			rnd	begin	end	f.a	rnd	begin	end	f.a	rnd	begin	end	f.a	rnd	begin	end	f.a	rnd	begin	end	f.a
	q	t	q	q	q	t	q	q	q	t	q	q	q	t	q	q	q	t	q	q	q	t
eil76	8,93%	0,8	2,46%	2,13%	1,25%	1,2	1,80%	0,99%	0,96%	1,5	1,62%	0,92%	0,77%	1,8	1,58%	0,44%	0,22%	2,2	1,14%	0,37%	0,18%	2,6
pr76	5,39%	0,9	0,60%	0,41%	0,34%	1,3	0,32%	0,24%	0,19%	1,7	0,25%	0,17%	0,10%	2,1	0,20%	0,16%	0,09%	2,4	0,17%	0,11%	0,07%	2,7
gr96	6,46%	1,7	1,68%	0,78%	0,70%	2,3	1,37%	0,59%	0,55%	2,9	0,94%	0,59%	0,55%	3,6	0,78%	0,59%	0,55%	4,2	0,82%	0,55%	0,47%	4,9
rat99	6,14%	1,9	2,53%	1,82%	1,62%	2,6	2,67%	0,90%	0,71%	3,3	2,81%	0,62%	0,56%	4,1	2,13%	0,53%	0,39%	5,2	1,28%	0,43%	0,38%	6,3
kroA100	6,67%	0,6	1,09%	0,73%	0,38%	0,9	0,84%	0,38%	0,33%	1,2	0,19%	0,22%	0,12%	1,5	0,18%	0,08%	0,03%	1,8	0,16%	0,03%	0,02%	2,1
kroB100	7,02%	0,8	1,61%	1,15%	1,02%	1,3	1,26%	0,70%	0,53%	1,8	0,72%	0,70%	0,42%	2,3	0,71%	0,47%	0,35%	2,8	0,76%	0,40%	0,38%	3,3
kroC100	6,61%	0,7	2,20%	1,05%	0,99%	1,2	1,19%	0,90%	0,75%	1,6	0,97%	0,63%	0,52%	2,1	0,79%	0,44%	0,37%	2,5	0,74%	0,38%	0,36%	2,9
kroD100	7,67%	0,8	2,20%	1,87%	2,11%	1,3	2,39%	2,02%	1,47%	1,8	1,44%	1,17%	0,97%	2,3	1,26%	0,89%	0,67%	2,8	0,97%	0,54%	0,46%	3,3
lin105	8,54%	0,5	1,50%	1,11%	1,19%	0,9	0,89%	0,70%	0,50%	1,4	0,83%	0,40%	0,41%	1,8	0,71%	0,37%	0,29%	2,2	0,46%	0,23%	0,23%	2,6
ch150	8,69%	5,4	2,94%	2,52%	2,34%	6,2	2,17%	1,89%	1,83%	6,9	1,77%	1,58%	1,37%	7,8	1,63%	1,46%	1,36%	8,7	1,31%	1,19%	0,92%	9,6
*pr439	10,45%	3,7	4,92%	4,35%	3,48%	11	4,59%	3,43%	2,96%	18	4,04%	3,16%	2,81%	25	3,34%	2,92%	2,56%	36,8	3,62%	3,13%	2,45%	45
Name	GGAemx		90				80				70				60							
			rnd	begin	end	f.a	rnd	begin	end	f.a	rnd	begin	end	f.a	rnd	begin	end	f.a				
	q	t	q	q	q	t	q	q	q	t	q	q	q	t	q	q	q	t				
eil76	0,04%	4,5	0,15%	0,04%	0,04%	4,1	0,18%	0,07%	0,04%	3,7	0,44%	0,15%	0,11%	3,3	1,07%	0,22%	0,11%	2,9				
pr76	0,04%	4,1	0,07%	0,04%	0,04%	3,8	0,12%	0,10%	0,05%	3,5	0,11%	0,10%	0,06%	3,2	0,15%	0,11%	0,06%	2,9				
gr96	0,12%	8,4	0,23%	0,16%	0,12%	7,7	0,27%	0,23%	0,20%	7	0,39%	0,35%	0,27%	6,3	0,74%	0,35%	0,31%	5,6				
rat99	0,00%	11,9	0,07%	0,00%	0,00%	10,8	0,56%	0,05%	0,02%	9,7	0,61%	0,16%	0,05%	8,5	1,13%	0,30%	0,25%	7,4				
kroA100	0,00%	3,6	0,00%	0,00%	0,00%	3,3	0,00%	0,00%	0,00%	3	0,01%	0,00%	0,00%	2,7	0,05%	0,00%	0,00%	2,4				
kroB100	0,10%	5,8	0,22%	0,12%	0,09%	5,3	0,31%	0,37%	0,22%	4,9	0,52%	0,20%	0,24%	4,3	0,81%	0,30%	0,29%	3,7				
kroC100	0,23%	5,3	0,37%	0,32%	0,23%	4,8	0,57%	0,56%	0,22%	4,3	0,52%	0,34%	0,29%	3,7	0,55%	0,32%	0,32%	3,3				
kroD100	0,08%	5,6	0,32%	0,28%	0,08%	5,2	0,58%	0,31%	0,26%	4,7	0,61%	0,45%	0,40%	4,3	0,88%	0,53%	0,45%	3,8				
lin105	0,10%	4,6	0,12%	0,09%	0,10%	4,2	0,11%	0,15%	0,10%	3,8	0,13%	0,12%	0,09%	3,4	0,21%	0,17%	0,12%	3,0				
ch150	0,30%	15,2	0,81%	0,35%	0,30%	14,1	0,86%	0,42%	0,37%	13	0,96%	0,69%	0,54%	12	1,16%	0,97%	0,84%	10,7				
*pr439	1,30%	91,8	1,72%	1,66%	1,34%	78,9	2,28%	2,14%	1,97%	70	2,78%	2,18%	2,03%	62	3,26%	2,91%	2,31%	52,4				

Figure 2
The running time as a function of amount of hybridization.



Source: Author's illustration

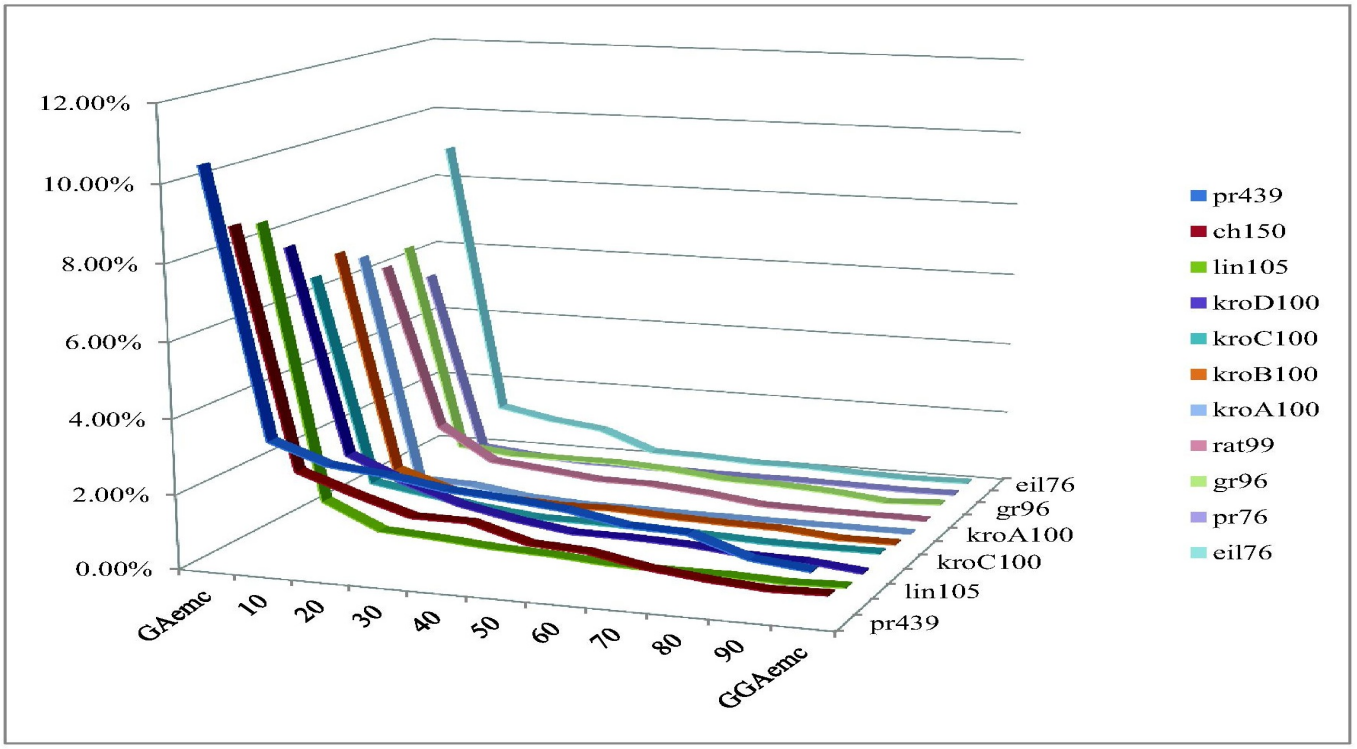
Figure 3
Quality results for case *pr439* as the amount of hybridization increases.



Source: Author's illustration

Similarly the quality of solution also improves as we increase the hybridization. In Figure 3 we see that for the case *pr439* with 439 cities the quality of solution improved from over 10% at no hybridization to approximately 3% at half hybridization and to 1.3% off the optimal at total hybridization. Similar behaviour can be observed at other cases (cf. Figure 4).

Figure 4
Quality results for all cases using an end-hybridization.



Source: Author's illustration

Note, that even small hybridization of 10% drastically improves solution — e.g. for the *pr439* case to only about 4% off the optimal. On the other hand, further hybridization keeps improving the result and it is up to the user to decide how much hybridization she wants to employ.

Probably the decision on the amount of hybridization should be made considering the running time. As seen in Figure 2 the number of cities increases the steepness of the function. Therefore high hybridization at large cases would probably increase the running time substantially. However, from our experiments seems to follow that also lower amounts of hybridization give satisfactory results (see Figure 4).

Conclusion

In the paper we studied the hybridization of a genetic algorithm with a simple local optimization algorithm. We showed that even a small hybridization substantially improves the quality of the result. Moreover, the hybridization in fact does not deteriorate the running time too much.

Our experiments further show that the best results are obtained when hybridization occurs in the last generations of the genetic algorithms. This seems to be in line with classical meta-heuristic algorithms like simulated annealing, which stop their diversification in the last iterations.

The paper opens a number of interesting questions for future research. The first one is related to the size of the problem. Namely, how does our GGA behave in cases, where Concorde can no longer compute the optimal solution? Next, can we use some other local optimization techniques instead of 2-opt and how would our GGA behave then? A very interesting question is also how to apply our technique to other NP-hard problems (e.g. 3-SAT or CLIQUE).

References

1. Applegate, D., Bixby, R., Chvátal, V., Cook, W. (2001), "TSP cuts which do not conform to the template paradigm", in Jünger, M., Naddef, D., (Eds), Computational Combinatorial Optimization, Springer, London, pp. 261-303.
2. Bosman, P., Thierens, D. (1999), "On the modelling of evolutionary algorithms", in Postma, E., Gyssens, M. (Eds), Proceedings of the 11th Belgium-Netherlands Conference on Artificial Intelligence, BNAIC, Vaeshartelt, pp. 67-74.
3. Djordjevic, M., Brodnik, A. (2011), "Quantitative analysis of separate and combined performance of local searcher and genetic algorithm", in Proceedings of the 33rd International Conference on Information Technology Interfaces, ITI2011, Faculty of Mathematics, Natural Sciences & Information Technologies, University of Primorska, Koper, pp. 515-520.
4. Djordjevic, M., Tuba, M., Djordjevic, B. (2009), "Impact of Grafting a 2-opt Algorithm Based Local Searcher Into the Genetic Algorithm", in Proceedings of the 9th WSEAS international conference on Applied informatics and communications, World Scientific and Engineering Academy and Society (WSEAS), Stevens Point, pp. 485-490.
5. Dorigo, M., Gambardella, L. M. (1997), "Ant colony system: A cooperative learning approach to the traveling salesman problem", IEEE Transactions on Evolutionary Computation, Vol. 1, No. 1, pp. 53-66.
6. Engels, C., Manthey, B. (2009), "Average-case approximation ratio of the 2-opt algorithm for the TSP", Operations Research Letters, Vol. 37, No. 2, pp. 83-84.
7. Freisleben, B., Merz, P. (1996), "New genetic local search operators for the traveling salesman problem", in Proceedings of the 4th International Conference on Parallel Problem Solving from Nature, Springer, London, pp. 890-899.
8. Garey, M. R., Johnson, D. S. (1979). Computers and Intractability: A Guide to the Theory of NP-completeness, New York: WH Freeman & Co.
9. Gutin, G., Punnen, A. P. (2002). The Traveling Salesman Problem and Its Variations, Dordrecht: Kluwer Academic Publishers.
10. Hahsler, M., Hornik, K. (2007), "TSP-Infrastructure for the traveling salesperson problem", Journal of Statistical Software, Vol. 23, No. 2, pp. 1-21.
11. Haupt, R. L., Haupt, S. E. (2004). Practical Genetic Algorithms, 2nd ed., New York: John Wiley & Sons.
12. Helsgaun, K. (2000), "An effective implementation of the Lin-Kernighan traveling salesman heuristic", European Journal of Operational Research, Vol. 126, No. 1, pp. 106-130.
13. Holland, J. (1975). Adaptation in natural and artificial systems, Ann Arbor: The University of Michigan Press.
14. Hoos, H. H., Stützle, T. (2005). Stochastic local search: Foundations and applications, Waltham: Morgan Kaufmann.

15. Merz, P., Freisleben, B. (2001), "Memetic algorithms for the traveling salesman problem", Complex Systems, Vol. 13, No. 4, pp. 297-345.
16. Reinelt, G. (1991), "TSPLIB – A traveling salesman problem library", ORSA Journal on Computing, Vol. 3, No. 4, pp. 376-384.
17. Sels, V., Vanhoucke, M. (2011), "A hybrid dual-population genetic algorithm for the single machine maximum lateness problem", in Merz, P., Hao, J. K., (Eds), Evolutionary Computation in Combinatorial Optimization, Springer, Berlin, pp. 14-25.

About the authors

Milan Djordjevic is a PhD student in Computer Science at the Faculty of Mathematics, Natural Sciences and Information Technologies in Koper, Slovenia. Mine research interests lie in theoretical computer science and operational research, particularly the combinatorial optimization and meta-heuristic algorithms. Author can be contacted at ***milan.djordjevic@student.upr.si***

Marko Grgurovič is a Master's student in Computer Science at the Faculty of Mathematics, Natural Sciences and Information Technologies in Koper, Slovenia. His research interests lie in theoretical computer science, particularly the design and analysis of algorithms. Author can be contacted at ***marko.grgurovic@student.upr.si***

Andrej Brodnik got his PhD from the University of Waterloo, Ontario, Canada. In 2002 he moved to University of Primorska. During the same time he also worked as a researcher and adjoined professor with the University of Technology in Luleå, Sweden. Andrej authored several tens of various scientific papers. He is also author and co-author of patents in Sweden and USA. The CiteSeer and ACM Digital Library lists over 200 citations of his works. Currently Prof. Brodnik holds positions with the University of Ljubljana and the University of Primorska. Author can be contacted at ***andrej.brodnik@upr.si***