Performance Analysis of Partial Use of Local Optimisation Operator on Genetic Algorithm for TSP

Milan Djordjevic, Marko Grgurovic, and Andrej Brodnik

Department of Information Science and Technology
University of Primorska, Koper, Slovenia
milan.djordjevic@student.upr.si, andrej.brodnik@upr.si,
marko.grgurovic@student.upr.si

Abstract. In this paper we study the influence of hybridization of a genetic algorithm with a local optimizer on instances of a Traveling Salesman Problem from a TSPLIB. In tests we applied hybridization at various percentages of genetic algorithm iterations. On one side 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 side the quality of solution on average deteriorated only from 0.21% till 1.40% worse than the optimal solution. We also studied at which iterations of the genetic algorithm to apply the hybridization. We applied it at random iterations, at the initial iterations, and the ending ones where the later proved to be the best.

1 Introduction

Genetic Algorithms (GA) use some mechanisms inspired by biological evolution [9]. They are applied on a finite set of individuals called population. Each individual in a population represents one of the feasible solutions of the 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 basic 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 population‘s next generation. The recombination operator combines parts of genetic code of the individuals (parents) into codes of new individuals (offsprings).
The components of the genetic algorithm software system are: Genotype, Fitness function, Recombinator, Selector, Mater, Replacer, Terminator, and in our system a Local searcher which is new extended component.
In this paper we study a well defined problem of a Traveling Salesman Problem (TSP). In the TSP a set $\{C_1, C_2, ...C_N\}$ of cities is considered

and 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

$$\sum_{i=1}^{N-1} d(C_{\pi(i)}, C_{\pi(i+1)}) + d(C_{\pi(N)}, C_{\pi(1)}). \quad (1)$$

This quantity is referred to as the tour length since it is the length of the tour a salesman would make when visiting the cities in the order specified by the permutation, returning at the end to the initial city. We will concentrate in this paper on the symmetric TSP in which the distances satisfy $d(C_i, C_j) = d(C_j, C_i)$ for $1 \leq i, j \leq N$ and more specificaly to the Euclidean distance. While the TSP is known to be *NP-hard* [7] even under substantial restrictions, the case with Euclidean distance is well researched and there are many excellent algorithms which perform well even on very large cases [7].
The 2-opt is a simple local search algorithm for the TSP. The main idea behind it is to take a route that crosses itself and reorder it so that it does not cross itself any more. The 2-opt local search will be used to hybridize GA metaheuristic to solve TSP. Although the 2-opt algorithm [5] performs well and can be applied to Traveling Salesman Problems with many cities, it finds only a local minimum. The nearest neighbour algorithm [10] is one of the most intuitive heuristic algorithms for the TSP. It's a greedy method for solving the TSP. The genetic algorithm considered in this paper are hybrid genetic algorithms, incorporating local search which have been referred to as Memetic Algorithms (MA) [11]. One example of hybridisation of genetic algorithms is shown in [12]

2 Grafted GA for the TSP

Grafting in botanic is when the tissues of one plant are affixed to the tissues of another. Grafting can reduce the time to flowering and shorten the breeding program. Local Searcher is an extension of the conventional genetic algorithm as it does not need to make use of genetic components. It facilitates the optimization of individual genomes outside the evolution process. There are many implementations of local searchers [6], some even in hardware [10].
In our algorithm, the pseudocode can be seen in Algorithm 1, after the recombination has been applied (line 7 in the pseudocode), a Local Searcher is used to optimize every single offspring genome (line 8 in the pseudocode). Because of the usage of such external optimizer the genetic algorithm is no longer *pure* and therefore we speak of a grafted genetic algorithm [2, 3]. This form of optimization is of a local kind. It alters the genome by heuristically changing the solution. Edge map crossover [6] is an implementation of the recombination operator (line 7 in the Algorithm 1). It makes use of a so called edge map. Distance preserving crossover [8] is another implementation of the recombination operator (line 7 in the Algorithm 1). It attempts to create a new tour with the same distance to both parents.
The local learcher is an extension to the conventional genetic algorithm as it needs not make use of genetic operators. It facilitates the optimization

of individual genomes outside the evolution process. The Local Searcher has no further knowledge on the execution of the genetic algorithm in the larger setting. The system will provide it with the genome it needs to locally optimize when needed.

Algorithm 1 Pseudocode

```
1: $t = 0$
2: initialize $(P(t))$
3: evaluate $(P(t))$
4: **while** not terminate $(P(t))$ **do**
5:     $sel$ = select $(P(t))$
6:     $mat$ = mate $(sel)$
7:     $rec$ = for each mated collection $m \in mat$ do recombination$(r)$
8:     $loc$ = for each genome $g$ in each recombined collection $r \in rec$ do local search
9:     $rep$ = replace$(loc, P(t))$
10:    $P(t+1)$ = select$(rep)$
11:    evaluate $(P(t+1))$
12:    $t = t + 1$
13: **end while**
```

The 2-opt local searcher is a local optimizer for the TSP that has been grafted into the standard genetic algorithm (line 8 in the Algorithm 1). This local optimizer performs the 2-opt heuristic that exchanges edges to reduce the length of a tour. An exchange step consists of removing two edges from the current tour and reconnecting the resulting two paths in the best possible way Fig. 1.

3 Experiment

For testing our strategy and comparing it to other solutions we used the instances of symmetric traveling salesman problem found on TSPLIB. We used relatively small instances, for which best solutions are known.

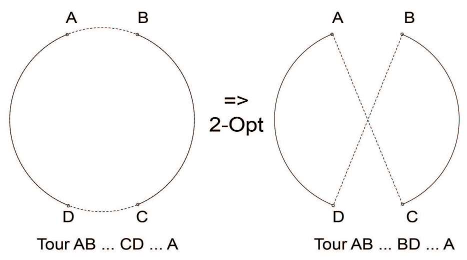

Fig. 1. Exchange step of 2-opt algorithm

The goal of this research was not to find a better algorithm, but rather to study on a controlled environment the impact of grafting a 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 (grafted genetic algorithm (GGA)) using two different recombination operators: an edge map crossover (GGAemc) and a distance preserving crossover (GGAdpc). As the upper and lower limits on the quality of solution we used greedy heuristic and Concorde [1] 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 in 10, 20, 30, 40, 50, 60, 70, 80 and 90 percents of the 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(R) 2.8 GHz CPU and Windows 7 operating system. In our results we can not cross 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* [4], while *Concorde* is an *AnsiC* application. However, we can compare running times of GGA for different instances and cases explained before.

4 Results

The results of an experiment are summarized in Table 1. Twenty cases from the well known TSPLIB were used for testing. The names of these cases are in the first column and the name always contains the size of the problem, i.e. the number of cities (which are between 14 and 150).
The last two columns are exact solutions (global minima) obtained by Concorde [1], together with execution times. A well known problem with moderate sized examples and tools to get optimal solutions were selected, recall that a goal of this research is not to improve solutions for difficult problems but to compare and quantitatively examine the effects of grafting local searches (in this case 2-opt based) to standard genetic algorithm. Such knowledge can be used to fine tune and calibrate a hybrid system which can then be used on large cases. These last two columns are used as a reference for all other tests. The second column in Table 1 represents lower bound for the quality of solution. It is a simple nearest neighbour heuristic. It is fast, but very unsophisticated and any reasonable algorithm should do better than that. This greedy heuristic gives results that are about 15% (except for some very small cases) worse than the optimal solution. The column titled quality shows by how many percent is the solution produced by this algorithm worse than the optimal solution. 0% in this column means that algorithm have found the best solution. The running times of the algorithm are from 0.2 to 2.3 in seconds.

Table 1. Five techniques for solving Euclidean TSP

Name	**Greedy**	**2-opt**	**GAemc**			**GAdpc**			**GGAemc**			**GGAdpc**			**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

The third column in the Table 1 corresponds to the pure 2-opt algorithm. As expected, it also gives quick but far from optimal solutions. It quickly finds a local minimum, but it is unable to broaden the search to find another local minimum. However, 2-opt algorithm is superior to greedy algorithm, the quality of the solution, with the similar running times from 0.2 to 2.5 seconds, is on average about 8% worse than optimal.

The fourth column in the Table 1 corresponds to the pure Genetic Algorithm. The running time, as expected, is significantly increased. While our GGA algorithm reached optimal solution below one second or few seconds (0.6 to 15.3 seconds), the running time for pure genetic algorithm was from 3.4 seconds to 100 seconds which was time-limit. In 12 out of 20 cases no optimal solution was found within that time limit, but in 8 cases an optimal solution was found and the middle column indicates in which generation that happened. For 12 cases where optimal solution was not found, the quality of found solution is expressed as for previous cases in percents above the optimal solution. The sixth column in Table 1 describes results obtained by our grafted algorithm, which is programmed with edge map crossover as recombination operator (GGAemc). In 17 out of 20 considered cases an optimal solution was found. Remaining three instances differ from optimal solution in 0.01, 0.10 and 0.22 percent. The solutions were found in relatively few generations and very fast. Execution times were 0.6 to 15.2 seconds.

The seventh column in Table 1 corresponds to our grafted genetic algorithm which contains a distance preserving crossover as recombination operator (GGAdpc). In 11 out of 20 considered cases an optimal solution was found. In remained 9 cases, delivered solutions differ from optimal in range from 0.13 to 0.32 percent. The running time and number of generations of GGAdpc, in comparison with GGAemc, are slightly lesser, particularly in the lowermost part of the table which represents more complex instances. Quantitative results on test cases from *TSPLIB* show that grafted algorithms, GGAemc and GGAdpc, have advantages. Even when their's components have serious drawbacks, their grafted combinations exhibits a very good behaviour. Results on examples from *TSPLIB* show that this grafted method combines good qualities from both methods applied and significantly outperforms each individual method.

The results of experiment extension are summarized in Table 2 and in Figures 2, 3 and 4.

The first column in Table 2 corresponds to the names of instances and the size of the problem (which are between 76 and 439) which are duplicated for better visualization of the table. The second column in the Table 2 presents the result of the pure Genetic Algorithm and Grafted Genetic Algorithm, both with edge map crossover as recombination operator (GAemc and GGAemc). This algorithms contain 0% and 100% frequency of local search, respectively. The q in Table 2 stands for *quality* differ from optimal solution. The t stands for running time.

The third column in Table 2 represents results for 10% and 90% frequency. The subcolumn titles *rnd, begin, end* stands for three variations of partial hybridization, random, begin sequence and end sequence, respectively. The result for 10% frequency shows that this kind of algorithm

Table 2. Partial Grafting of a Genetic Algorithm

Name	GAemc		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

	GGAemc		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

settings is fast (the time vary from 0.9 seconds to 11.0 seconds), but the quality of solution (which vary from 0.34% to 4.92%), are weak.
The best performance of all tested cases was achieved in the configuration with 90% frequency, especially in variation with end sequence, with results coloured in red. In 9 out of 11 tested instances the results was the same as for GGAemc. For instance (kroB100) result was better in 0.01%, and for instance pr439 a result is worse in 0.04%. Furthermore, the solutions provided by, 90% end sequnce variations, were found faster then by GGAemc. The running time vary from 3.3 to 78.9 seconds, compared to the time achieved by GGAemc, which vary from 3.6 to 91.8 seconds. This mean, that the *same quality* of result was achieved in *shorter time*. The results for instance pr439 can be seen in Figure 2. In 95% of all tested cases, (look in columns from 3 to 7 in Table 2) the best performance was achieved in variation with end sequence. Additionally, the results of end sequence for all instances can be seen in Figure 4. In all three variations of hybridization (random, begin and end) the running time is the same for particular frequency. For all tested instances the running time grows almost linerly with regard to percent of hybridization, see Figure 3.

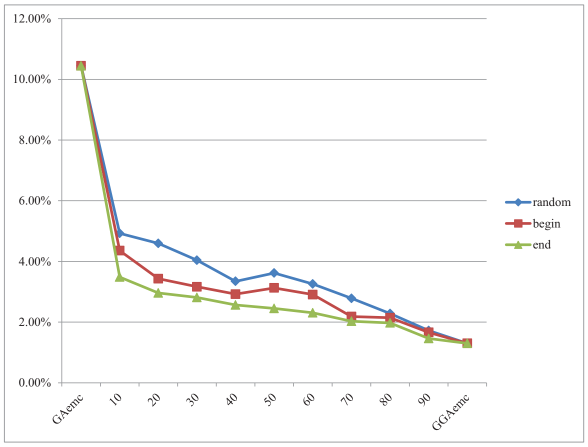

Fig. 2. Results for pr439

5 Conclusions

The goal of this paper was to investigate influence of grafting a 2-opt based local searcher into the standard genetic algorithm, for solving the Travelling Salesman Problem with Euclidean distance. It is known that genetic algorithms are very successful when implemented for many NP-hard problems. However, they are much more effective if some specific knowledge about particular problem is utilized. In our first experiment we compared two direct techniques, with our grafted genetic algorithms.

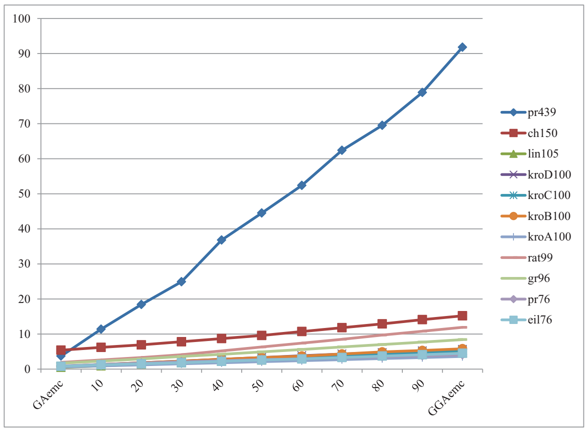

Fig. 3. Running times

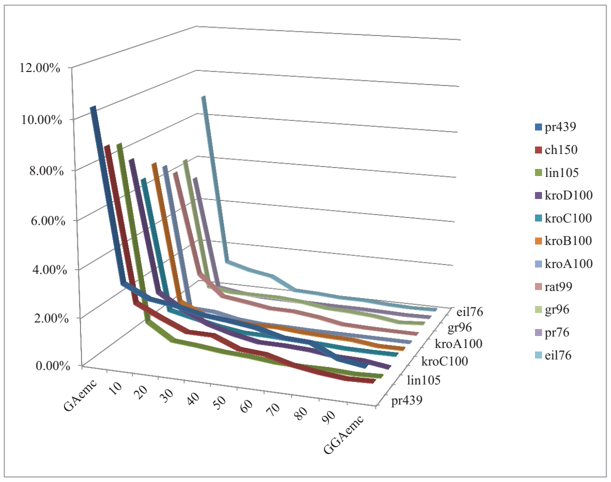

Fig. 4. Results for end sequence

Solutions from Concorde and greedy algorithm were added for better comparison. Quantitative results on test cases from *TSPLIB* show that grafted algorithms have advantages. Even when both components have serious drawbacks, their grafted combinations exhibits a very good behaviour. Results on examples from *TSPLIB* show that this method combines good qualities from both methods applied and significantly outperforms each individual method.

In the second part of an experiment an influence of partial grafting a 2-opt local searcher into genetic algorithm was studied. The *best performance* was achieved in a configuration with 90% frequency with end sequence. In a comparison with a performance of GGAemc the *same quality* of results was achieved in a *shorter time*, on average a 7% of running time was spared. The cases with 10% frequency use of local search provides fast and far from optimal solutions but still better then the GAemc, with small increase in time. The configurations with 50% frequency use of local searcher present a good examples of trade-off between a running time and quality, especcialy in setting with ending sequence of local searcher. From the results obtained in Table 2, we can conclude that the best gain is attained when a local searcher is used in an ending sequence of the algorithm and in frequency not less then 50% and not more than 90%. There are several issues for future research, such as investigating the effects of a different use of the local optimisation and other metaheuristic algorithms, analyzing the individual performance gains provided by the local search, and to look at how to scale up the algorithm for solving large instances of TSP.

References

1. Applegate, D., Bixby, R., Chvátal, V., Cook, W.: TSP cuts which do not conform to the template paradigm. In: Computational Combinatorial Optimization (2001) 261–304
2. Djordjevic, M.: Influence of Grafting a Hybrid Searcher Into the Evolutionary Algorithm. In: Proceedings of the Seventeenth International Electrotechnical and Computer Science Conference. Slovenian Section IEEE (2008) 115–118
3. Djordjevic, M., Tuba, M., Djordjevic, B.: 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) (2009) 485–490
4. Bosman, P., Thierens, D.: On the modelling of evolutionary algorithms. In: Proceedings of the Eleventh Belgium-Netherlands Conference on Artificial Intelligence BNAIC (1999) 67–74
5. Engels, C., Manthey, B.: Average-case approximation ratio of the 2-opt algorithm for the TSP. Operations Research Letters **37** (2009) 83–84
6. Freisleben, B., Merz, P.: New genetic local search operators for the traveling salesman problem. In: Proceedings of the 4th International Conference on Parallel Problem Solving from Nature (1996) 890–899

7. Garey, M., Johnson, D.: Computers and Intractability: A Guide to the Theory of NP-completeness. WH Freeman & Co, New York (1979)
8. Helsgaun, K.: An effective implementation of the Lin-Kernighan traveling salesman heuristic. European Journal of Operational Research **126** (2000) 106–130
9. Holland, J.: Adaptation in natural and artificial systems. The University of Michigan Press, Ann Arbor (1975)
10. Hoos, H., Stutzle, T.: Stochastic local search: Foundations and applications. Morgan Kaufmann (2005)
11. Merz, P., Freisleben, B.: Memetic algorithms for the traveling salesman problem. Complex Systems **13** (2001) 297–346
12. Sels, V., Vanhoucke, M.: A hybrid dual-population genetic algorithm for the single machine maximum lateness problem. In: Proceedings of the 11th European conference on Evolutionary computation in combinatorial optimization (2011) 14–25