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

**Abstract**

In this paper we study the influence of hybridization of a genetic algorithm with a local optimizer on solving instances of a Traveling Salesman Problem from 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.

**Keywords:** genetic algorithm, traveling salesman problem, optimization, grafted genetic algorithm

**JEL classification:** C61 - Optimization Techniques; Programming Models; Dynamic Analysis

**Introduction**

Genetic Algorithms (GA) use some mechanisms inspired by biological evolution (Holland, 1975). 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 next generation. The recombination operator combines parts of genetic code of the individuals (parents) into codes of new individuals (offspring).

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 a new extended component.

In this paper we study a well-defined problem of a Traveling Salesman Problem (TSP). In the TSP a set *{C1, C2, ..., CN}* of cities is considered and for each pair *{Ci, Cj}* of distinct cities a distance *d(Ci ,Cj)* is given. The goal is to find an ordering *π* of the cities that minimizes the quantity.

$$\sum\_{i=1}^{N-1}d\left(C\_{π\left(i\right)}, C\_{π\left(i+1\right)}\right) + d\left(C\_{π\left(n\right)}, C\_{π\left(1\right)}\right).$$

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(Ci, Cj)=d(Cj,Ci)* for *1 ≤ i, j ≤ N* and more specifically to the Euclidean distance. The TSP is known to be NP-hard (Garey and Johnson, 1979) even under substantial restrictions. The case with Euclidean distance is well researched and there are algorithms which perform well even on very large cases (Applegate, Bixby, Chvátal and Cook, 2001).

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 meta-heuristic to solve TSP. Although the 2-opt algorithm (Engels and Manthey, 2009) performs well and can be applied to TSP with many cities, it finds only a local minimum. The nearest neighbor algorithm (Hoos and Stutzle, 2005) is one of the most intuitive heuristic algorithms for the TSP. It is a greedy method for solving the TSP. The genetic algorithms considered in this paper are hybrid genetic algorithms, incorporating local search (cf. Memetic Algorithms, (Merz and Freisleben, 2001)). One example of hybridization of genetic algorithms is shown in (Sels and Vanhoucke, 2011).

**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. 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 Stutzle, 2005).

In our algorithm, we did not use a k-opt heuristics due to its complexity, but rather its simpler version 2-opt explained in the previous section. The hybridization occurs in line 8 of the pseudocode of our algorithm given lower. Because of use of local optimizer the genetic algorithm is no longer *canonical* and we speak of a grafted genetic algorithm – GGA (Djordjevic and Brodnik, 2011), (Djordjevic M., Tuba and Djordjevic B., 2009).

Grafted Genetic Algorithm:

1. *t=0*
2. *p(t):=Initialize()*
3. *q(t):= Evaluate(P(t))*
4. **While** *(t < t\_max)* **and** *(q(t) < q\_expected)*
5. *sel:= Select(P(t))*
6. *mat:= Mate(sel)*
7. *rec:=* **for each** pair *m* ∈ *mat* **do** *Recombine(m)*
8. *loc:=* **for each** genome *r* ∈ *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**

emc je malo boljši od dpc glede na kakovost rešitve. za obe rekombinaciji smo se odločili, da preverimo, if influence of hybridization on GA is recombination dependent. (see higher). to results.

MILAN:: how emc and dpc work.

can be seen below, after the recombination has been applied (line 7 in the pseudocode). Edge map crossover (Merz and Freisleben, 2001) is an implementation of the recombination operator (line 7 in the algorithm). It makes use of a so called edge map. Distance preserving crossover (Helsgaun, 2000) is another implementation of the recombination operator (line 7 in the algorithm). It attempts to create a new tour with the same distance to both parents.

The local searcher 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.

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). 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, as shown in Figure 1.

*Figure 1*

2-Opt

[2opt.jpg]

*Source:* Author’s illustration

**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. 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 (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 (Applegate, Bixby, Chvátal and Cook, 2001) 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 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 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 before.

*Table 1*

Five Techniques for Solving Euclidean TSP

[table1.jpg]

**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 sites, 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

$q\_{A}=\frac{l\_{A}-l\_{C}}{l\_{C}}$ *(2)*

where *lC* is a path length obtained by Concorde and *lA* 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 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 sites 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 GGAemc and for GGdpc 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 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 GGAemc, 30 times increase for GGAdpc and even 70 times increase for Concorde.

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 (GGAemc) and only eleven cases from TSPLIB (cf. Table 2).

*Table 2*

Partial Grafting of a Genetic Algorithm

[table2.jpg]

In the experiment we were increasing the number of generations in which we applied hybridization by 10 percents: from 0% — column *GAemc* in Table 2 till 100% — column *GGAemc*. Moreover, we also varied the generations in which we applied the hybridization: either in random generations (column *rnd*), in the beginning ones (*begin*) or in 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 more diversification. 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).

*Figure 2*

The Running Time as a Function of Amount of Hybridization

[time.jpg]

*Source:* Author’s illustration

*Figure 3*

Quality Results for Case pr439 as the Amount of Hybridization Increases

[pr439.jpg]

*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 sites 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 behavior can be observed at other cases (cf. Figure 4).

*Figure 4*

Quality Results for All Cases Using End-Hybridization

[end.jpg]

*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 sites 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).

**Conclusions**

In the paper we hybridized a simple local optimization algorithm and genetic 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 GA. 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. The first one is related to the size of the problem. Namely, how do our GGA behave in cases, where Concorde can no more compute the optimal solution. Next, can we use some other local optimization technique instead of 2-opt and how would our GGA behave then. A very interesting question would also be how would our technique work in other NP-complete problems. For example in 3-SAT or CLIQUE.

**References**

Applegate, D., Bixby, R., Chvátal, V., Cook, W. (2001), “TSP cuts which do not conform to the template paradigm”, Computational Combinatorial Optimization, pp. 261-304.

Bosman, P., Thierens, D. (1999), “On the modelling of evolutionary algorithms”, in Proceedings of the Eleventh Belgium-Netherlands Conference on Artificial Intelligence, BNAIC pp. 67-74.

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 pp. 515-520.

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) pp. 485-490.

Engels, C., Manthey, B. (2009), “Average-case approximation ratio of the 2-opt algorithm for the TSP”, Operations Research Letters, Vol. 37, pp. 83-84.

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 pp. 890-899.

Garey, M., Johnson, D. (1979), Computers and Intractability: A Guide to the Theory of NP-completeness, WH Freeman & Co, New York.

Helsgaun, K. (2000), “An effective implementation of the Lin-Kernighan traveling salesman heuristic”, European Journal of Operational Research, Vol. 126, pp. 106-130.

Holland, J. (1975), Adaptation in natural and artificial systems, The University of Michigan Press, Ann Arbor.

Hoos, H., Stutzle, T. (2005), Stochastic local search: Foundations and applications. Morgan Kaufmann.

Merz, P., Freisleben, B. (2001), “Memetic algorithms for the traveling salesman problem”, Complex Systems, Vol. 13, pp. 297-346.

Sels, V., Vanhoucke, M. (2011), “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, pp. 14-25.