Genetic Local Search for the TSP: New Results

Peter Merz and Bernd Freisleben

Department of Electrical Engineering and Computer Science (FB 12)
University of Siegen
Hölderlinstr. 3, D–57068 Siegen, Germany
E-Mail: {pmerz,freisleb}@informatik.uni-siegen.de

Abstract— **The combination of local search heuristics and genetic algorithms has been shown to be an effective approach for finding near-optimum solutions to the traveling salesman problem. In this paper, previously proposed genetic local search algorithms for the symmetric and asymmetric traveling salesman problem are revisited and potential improvements are identified. Since local search is the central component in which most of the computation time is spent, improving the efficiency of the local search operators is crucial for improving the overall performance of the algorithms. The modifications of the algorithms are described and the new results obtained are presented. The results indicate that the improved algorithms are able to arrive at better solutions in significantly less time.**

I. Introduction

Consider a salesman who wants to start from his home city, visit each of a set of n cities exactly once, and then return home. Since the salesman is interested in finding the shortest possible route, this problem corresponds to finding a shortest hamiltonian cycle in a complete graph $G = (V, E)$ of n nodes. Thus, the *traveling salesman problem* (TSP) [22], [32] consists of finding a permutation π of the set $\{1, 2, 3, \ldots, n\}$ that minimizes the quantity

$$C(\pi) = \sum_{i=1}^{n-1} d_{\pi(i),\pi(i+1)} + d_{\pi(n),\pi(1)} \qquad (1)$$

where d_{ij} denotes the distance between city i and j. In the *symmetric* TSP (STSP), $d_{ij} = d_{ji}$ holds for all i, j, while in the *asymmetric* TSP (ATSP) this condition is not satisfied.

The TSP is among the best–known combinatorial optimization problems. It has attracted many researchers from various fields, perhaps because it is easy to state but extremely hard to solve. Since it belongs to the class of $\mathcal{NP}$–hard problems [14], fast approximation techniques for finding optimum or near–optimum solutions to large TSP instances in a given time limit are required.

In addition to the classical heuristics developed especially for the TSP [32], there are several problem independent search algorithms which have been applied to the TSP, such as *ant colonies* [13], *genetic algorithms* [17], *local search* [20], *neural networks* [30], *simulated annealing* [21], and *tabu search* [9]. The results published in the literature indicate that it is necessary to combine some of these methods in order to arrive at high quality solutions, particularly for large problem instances. For example, local search has been used in [7], [8], [18], [33], [34] to improve genetic algorithms (GAs) for the TSP. As a consequence, Gorges-Schleuter and Mühlenbein [15], [16], [27] have proposed a GA where all individuals of the population are local minima with respect to the embedded local search method. Ulder et al. [35] compared this *genetic local search* (GLS) approach with other heuristics and observed that GLS is superior to simulated annealing as well as multi–start local search. In [11], [12] we proposed new operators for GLS which are designed to produce better (locally optimal) individuals from existing ones, with the ability to find optimal solutions for symmetric TSP instances of up to 1400 cities.

In this paper, the GLS algorithms for the symmetric and the asymmetric TSP proposed in [11], [12] are revisited. Since the central component of the algorithms is the local search operator, and improving the efficiency of this operator consequently improves the overall performance, we will take a closer look at the hill–climbers used in our approach and discuss how they can be tuned for speed as well as for quality. The results obtained indicate that compared to our winning algorithm of the *First International Contest on Evolutionary Optimization* held as part of the *1996 IEEE International Conference on Evolutionary Computation* in Nagoya, Japan [4], the computation times could be reduced significantly, and the solution qualities could further be improved.

The paper is organized as follows. Section 2 gives a general description of the GLS approach to the TSP. Section 3 presents genetic operators for the symmetric TSP, while section 4 is devoted to the asymmetric case. Section 5 summarizes the improvements made in our implementation. The computational results are presented in section 6. Section 7 concludes the paper and outlines areas for future research.

II. The GLS Algorithm for the TSP

In the GLS algorithm, all individuals represent local minima. Therefore, after applying a genetic operator, the local search procedure must be applied to the resulting individual. The GLS algorithm proposed to solve the TSP is presented in Fig. 1.

First, the initial population is created by a tour construction technique, which in our case is the nearest–neighbor heuristic [22]. Then, a suitable local search procedure is applied to each individual. In the main loop, crossover and mutation operators are applied to randomly selected individuals a predefined number of times. To achieve local optimality, the hill–climber is employed after each application of an operator, and the newly created individuals are

```
procedure GLS;
    begin
        initialize population P with a construction heuristic;
        foreach individual i ∈ P do i := Local-Search(i);
        repeat
            for i := 1 to #crossovers do
                begin
                    select two parents i_a, i_b ∈ P randomly;
                    i_c := DPX(i_a, i_b);
                    i_c := Local-Search(i_c);
                    add individual i_c to P;
                end;
            for i := 1 to #mutations do
                begin
                    select an individual i ∈ P randomly;
                    i_m := Mutate(i);
                    i_m := Local-Search(i_m);
                    add individual i_m to P;
                end;
            P := select(P);
        until P converged;
    end;
```

Fig. 1. The GLS Algorithm for the TSP

inserted into the population. In the last step of each generation, the population is reduced to its initial size by first eliminating old individuals which are similar to the newly created individuals, and then selecting the individuals with the highest fitness until the population size is reached.

Unlike classical genetic algorithms, where mutation is refered to as a background operator, GLS makes equal use of both crossover and mutation operators. The goal of the operators used in conjunction with the local search method is to produce hopefully better individuals from existing ones by utilizing the information already contained in the current population. The approach is motivated by the observation [5], [28] that near–optimum and optimal solutions are quite similar: they have many edges in common. To describe the similarity of solutions more precisely, we define the distance between tours corresponding to [6], [24]. Let $G = (V, E)$ be the graph of a given TSP instance and T_1, T_2 feasible tours, then the distance D between the tours is defined as

$$D(T_1, T_2) = |\{e \in E \mid e \in T_1 \wedge e \notin T_2\}|. \qquad (2)$$

Since optima and near–optimum local minima lie relatively close to each other in the search space, it seems to be reasonable to explore the regions around the best solutions collected so far during the search.

As described in [11], [12], crossover and mutation perform "jumps" in the search space, and it is difficult to find a diversification scheme that leads to promising regions of the search space without degrading to an uncontrolled random walk.

The crossover introduced in [11], [12] is called the *distance preserving crossover* (DPX). Here, a child i_c is created by copying the edges to the offspring that are found in both parents i_a and i_b, and reconnecting the resulting tour fragments without reinserting the edges that are different in the parents. Thus, the child has equal distance to both parents, and this distance is also equal to the distance between the parents themselves:

$$D(i_c, i_a) = D(i_c, i_b) = D(i_a, i_b). \qquad (3)$$

The DPX is motivated by the observation of Boese [5], that for the symmetric 532-cities instance of Padberg and Rinaldi [29] the optimum lies more or less in a single valley of near–optimum local minima and the average distance between these near–optimum solutions is similar to their distance to the optimum. Another motivation is that by identifying the edges that are not shared by the parents, the search will be focused on particular regions of the search space. By inheriting the shared edges, we assume that these edges are likely to be contained in the optimum as well, because they have been proven to be "building blocks" during the search. With these ideas in mind, the proposed crossover operator neither depends on a particular problem (as long as the distance criterion (3) can be fulfilled), nor on the type of local search subsequently used.

On the other hand, the mutation operator is highly depending on the hill–climber chosen. Since it is a unary operator, we must decide at a single point in the search space which region to explore next. It is reasonable to replace only a few edges in the individual in a way that the permutation will not be reversed by the subsequent local search. Therefore, some knowledge about the incorporated hill–climber might be helpful.

III. The Operators for the STSP

The local search method used for symmetric TSP instances is the variable k-change heuristic suggested by Lin and Kernighan [23]. In each step, a varying number of edges is exchanged, until no exchange will reduce the tour length further, i.e. when a local minimum is reached. This heuristic produces quite good results even when applied alone. For most instances defined in the TSPLIB [31], the average percentage excess is about 2% above the optimum, although there are instances with topographical structures for which the results are much worse, such as the instance `fl3795` [20].

The mutation operator used in the symmetric TSP is called the *non-sequential four-change* [23] which has also been used in the *large step Markov chain algorithm* proposed by Martin et al. [25] and the *iterated Lin-Kernighan heuristic* suggested by Johnson [19]. Since in the Lin-Kernighan heuristic only sequential changes are performed, the effects of this kind of mutation cannot be reversed in a single step, and there is a high probability for ending up with a new solution after the local search phase. Since only four edges are exchanged, most information coded in the genotype will be preserved.

IV. The Operators for the ATSP

In the ATSP, the distance value associated with edge (i, j) is different from the one of edge (j, i), such that local

search procedures developed for symmetric instances are not very effective. The *2-opt* algorithm, for instance, tries to find pairs of edges for which an exchange results in a reduction of the total tour length. For example, consider the individual A shown in Fig. 2, which represents a tour from city 5 to city 3 to city 7 and so on.

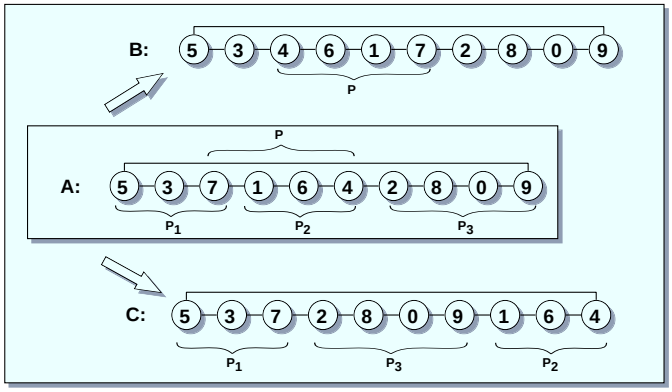

Fig. 2. Illustration of Inversion and Local Search Operators

If the exchange of the edges $(3,7)$ and $(4,2)$ by $(3,4)$ and $(7,2)$ will lead to a shorter tour, the *2-opt* algorithm will perform the swap by reversing the order of the subpath from city 7 to 4, as shown by the tour B in Fig. 2.

In the ATSP the reversal of the subpath leads to a k-change. In the above example, the distance between tour A and B is $D = 5$, because edges $(3,4), (4,6), (6,1), (1,7)$ and $(7,2)$ are not contained in tour A. It is easy to see that for the ATSP the minimum distance between tours is $D = 3$. Individual C in Fig. 2 represents a tour with a distance $D = 3$ to tour A. As illustrated, tour C can be obtained by swapping subpath P_2 and P_3 in tour A, which is equivalent to a *3-opt* move. Analogously, it is possible to perform a valid *4-opt* move by rearranging a tour that consists of subpaths $P_1P_2P_3P_4$ in a way that the order of the subpaths in the resulting tour is $P_1P_4P_3P_2$.

The local search procedure used in our GA for the ATSP performs both *3-opt* and *4-opt* moves. To reduce the running time for finding a *4-opt* move, information obtained from previous iterations of the search is used: the four edges to exchange in a step consist of a pair of edges identified as candidates in the current step and a pair of edges identified in a previous iteration. This enables us to perform 4-changes without increasing the running times compared to *3-opt* without additional 4-changes.

Mutation is performed by reversing a randomly chosen subpath of length 6 analogously to the example shown in Fig. 2. Thus, by the reversal of a subpath the mutation operator performs a random 7-change on the current tour.

V. Implementation Issues

The efficiency of the embedded local search algorithm has a large impact on the performance of the entire algorithm, since about 95% – 99% of the total CPU time is spent in the local search procedure. Thus, the main goal is to optimize the local search procedure in order to run much faster without losing too much quality. A complete enumeration of the neighborhood is not very effective, since the size of the *k-opt* neighborhood grows polynomially with k, i.e. the neighborhood defined as

$$\mathcal{N}_{kopt}(T) = \{T' \mid D(T,T') \leq k\} \tag{4}$$

is of size $O(n^k)$. Even for small k, the computation times may increase considerably. To obtain computation times that grow subquadratically in practice, a small portion of the neighborhood may be inspected where good solutions are most likely to be found. This can be achieved by maintaining a list of nearest neighbors for each node [20], [32]. If an edge in the current tour is considered for replacement, only candidates drawn from the nearest neighbor list are examined. If the size of the neighbor list is bound by a constant, it is possible to identify an improving *k-opt* move associated with a given starting node in constant time, and hence the time for checking an improving move is $O(n)$. Further reductions result from performing a fixed radius nearest neighbor search, as proposed by Bentley [1], [3].

Another mechanism to speed up local search algorithms is to incorporate a "don't look bit" for each node, in order to reduce the time for checking the interesting neighbors [1]. With the don't look bit set to 1, the node is not considered as a starting point for finding an improving move. Initially, all bits are set to 0. If an improving move could not be found starting at node i, the don't look bit for that node is set. On the other hand, the don't look bit will be cleared if an improving move has been found that inserts an edge incident to node i.

In our implementation, we use a nearest neighbor list of size $m = 100$ for each node, which is initialized by nearest neighbor queries on a two-dimensional binary search tree [2] in case of the algorithm for the STSP. In the ATSP, we adapted the recursive *quicksort* algorithm to sort the first m entries of the rows of the distance matrix. In both local search procedures, a data structure for maintaining don't look bits is incorporated, with the local search for the initial population starting with all don't look bits set to zero. After the crossover has been performed, only the don't look bits of the nodes that are incident to the edges not shared by both parents are cleared. Similarly, after mutation, only nodes incident to the edges newly included in the tour have their don't look flags set to zero. This focuses the search of the hill-climber to the promising regions of the search space and also reduces the time for checking the interesting members of the neighborhood.

Additionally, in the algorithm for the STSP, some changes have been made to deal with large instances of up to 100.000 cities. Since for large instances it is not possible to store the entire distance matrix in main memory, the Euclidean distances are computed online. Since this is a rather expensive operation, a distance cache of size $3 \cdot n$ is maintained, where the first n entries are used to cache the distances of the edges in the current tour and the remaining $2 \cdot n$ entries are organized as described in [2]. The average hit rate of the cache varies between 80% and 95%.

STSP Results				
instance	gen	best	average	t in s
d198	40	15780 (0.00 %)	15780.1 (0.00 %)	163
lin318	60	42029 (0.00 %)	42029.0 (0.00 %)	168
pcb442	140	50778 (0.00 %)	50778.0 (0.00 %)	178
att532	140	27686 (0.00 %)	27698.4 (0.04 %)	290
rat783	300	8806 (0.00 %)	8806.2 (0.00 %)	424
fl1577	300	22249 (0.20 %)	22250.5 (0.21 %)	12762
fl3795	400	28776 (0.18 %)	28868.5 (0.51 %)	17824

ATSP Results				
instance	gen	best	average	t in s
p43	60	5620 (0.00 %)	5620.1 (0.00 %)	11
ry48p	700	14422 (0.00 %)	14451.2 (0.20 %)	72
ft70	40	38673 (0.00 %)	38674.2 (0.00 %)	111
kro124p	200	36230 (0.00 %)	36231.5 (0.00 %)	35
ftv170	400	2755 (0.00 %)	2762.2 (0.26 %)	100

Fig. 3. The Computational Results

Another target for optimizations is the Lin-Kernighan heuristic itself. Most of the computation time is spent in submoves that will be reversed later in the algorithm. Hence, it is profitable to distinguish between tentative and permanent moves. Applegate and Cook have proposed a *segment tree* data structure for efficiently managing tentative moves, as described in [10]. Our approach is similar: instead of using a segment tree, we operate on a segment list that represents a tentative tour. Operations performing a flip on this tentative tour are highly optimized, such that a high performance gain compared to the simple array representation can be achieved. The running times for all operations are in $O(1)$, since the data structure is limited to perform 20 flips only. In practice, this has been proven to be sufficient.

VI. Results

The algorithms presented in this paper have been implemented in *C++* on a DEC Alphastation 255 running Digital Unix V4.

Detailed information about the development of the solution qualities in each of the experiments conducted is given in the appendix. Fig. 3 summarizes the results obtained for several STSP and ATSP instances taken from the TSPLIB [31]. In the figure, *gen* denotes the number of generations performed, *best* shows the length of the best tour found together with its deviation from the known optimum in percent, *average* displays the same information for the average of 20 runs, and *t in s* shows the average computation times (in seconds) for a single run on a DEC Alpha CPU with 233 MHz. For the instances `fl1577` and `fl3795`, we provide the deviation from the best known lower bound. For the STSP instances, a population size of $P = 20$ was used in our experiments, except for `d198` where the population contained $P = 10$ individuals. All runs on the asymmetric instances were performed with $P = 40$. For both the STSP and the ATSP, we used a mutation and a crossover rate of 0.5, i.e. P individuals were created in each iteration.

The computation times could be reduced significantly compared to the results in [11] for all STSP instances. Better qualities could be achieved for all instances except for instance `d198`. Obviously, the average quality (0.00 %) cannot be improved further.

For example, the average running times for the 783-cities problem `rat783` were reduced from 14880 seconds to 424 seconds (a factor of more than 35), while the quality of the results increased: the average percentage excess over the optimum has dropped from 0.04% to 0.0023%. Remarkable results have also been obtained for the smallest yet unsolved problem defined in the TSPLIB, `fl1577`. The average time for a local search could be reduced by a factor of 48 to 2.12 seconds. This enabled us to run the algorithm for an increased number of generations, ending up with a best final tour length of 22249, which is the best known upper bound for this instance. The gain in quality for the average final tour length is also worth mentioning: the percentage excess over the lower bound dropped from 0.46% to 0.21%.

The optimum for instance `fl3795` is known to lie in the interval $[28723, 28772]$. The best tour found by our algorithm for this instance has a tour length of 0.014% above the upper bound. The relatively high average tour length after 400 generations may be improved by running the algorithm for a longer time.

The reasons for the strong increases of the computation times for the instances `fl1577` and `fl3795` are not the problem sizes itself, but the characteristics of these instances. They are pathologically clustered, making it much harder to optimize them by neighborhood search. For example, for another problem of comparable size (`pcb3038`), the average running time for a local search is about 0.30 seconds and thus more than a factor of 7 faster than in the case of `fl3795` (2.22 seconds).

For the ATSP, the results in Fig. 3 represent a considerable performance gain in comparison to the results presented in [11]. For all instances except `ry48p`, the average quality could be improved. For the largest instance investigated, the average percentage excess over the optimum has dropped from 0.40% to 0.26% using less than half of the computation time. An even higher reduction of the computation time could be achieved for problem `ft70`, which is known to be a relatively hard problem for a hill–climber [26]. In this case, the new algorithm is about 5.75 times faster, and in almost all runs the optimum is found. With the discussed modifications, the running times for all ATSP instances are less than two minutes.

The improvements of our algorithms have led to much faster local search procedures which produce slightly worse tours than before. However, for both types of instances it has proven to be more effective to produce more generations in the GA than to use more powerful embedded local search methods.

VII. Conclusions

In this paper, improvements of GLS algorithms for the TSP have been presented. The results presented for sev-

eral symmetric and asymmetric TSP instances have shown that some fine tuning of the local search procedures in the GLS algorithms has a significant impact on the overall performance. Sophisticated data structures and hill–climbers with subquadratic runtime characteristics are the keys to efficient neighborhood searches, in particular if large problem instances are considered. The experiments conducted have demonstrated that a slightly weaker but considerably faster local search method has the advantage of getting executed more often in a given time, which leads to a more robust search algorithm that is able to carefully explore the search space.

There are several issues for future research, such as investigating the effects of a parallel implementation of the GLS approach, analyzing the individual performance gains provided by the crossover and mutation operators, and conducting further tests of the algorithms on other possibly more complex TSP instances.

References

[1] J. L. Bentley, "Experiments on Traveling Salesman Heuristics," in *Proceedings of the First Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 91–99, 1990.

[2] J. L. Bentley, "K-d-Trees for Semidynamic Point Sets," in *Proceedings of the Sixth Annual ACM Symposium on Computational Geometry*, pp. 187–197, 1990.

[3] J. L. Bentley, "Fast Algorithms for Geometric Traveling Salesman Problems," *ORSA Journal on Computing*, vol. 4, no. 4, pp. 387–411, 1992.

[4] H. Bersini, M. Dorigo, S. Langerman, G. Seront, and L. Gambardella, "Results of the First International Contest on Evolutionary Optimisation (1st ICEO)," in *Proceedings of the 1996 IEEE International Conference on Evolutionary Computation*, (Nagoya, Japan), pp. 611–615, 1996.

[5] K. Boese, "Cost versus Distance in the Traveling Salesman Problem," Tech. Rep. TR-950018, UCLA CS Department, 1995.

[6] K. Boese, A. Kahng, and S. Muddu, "A New Adaptive Multi-start Technique for Combinatorial Global Optimizations," *Operations Research Letters*, vol. 16, pp. 101–113, 1994.

[7] R. M. Brady, "Optimization Strategies Gleaned from Biological Evolution," *Nature*, vol. 317, pp. 804–806, 1985.

[8] T. G. Bui and B. R. Moon, "A New Genetic Approach for the Traveling Salesman Problem," in *Proceedings of the First IEEE Conference on Evolutionary Computation*, pp. 7–12, 1994.

[9] C.-N. Fiechter, "A Parallel Tabu Search Algorithm for Large Traveling Salesman Problems," *Discrete Applied Mathematics and Combinatorial Operations Research and Computer Science*, vol. 51, pp. 243–267, 1994.

[10] M. L. Fredman, D. S. Johnson, L. A. McGeoch, and G. Ostheimer, "Data Structures for Traveling Salesmen," *Journal of Algorithms*, vol. 18, pp. 432–479, 1995.

[11] B. Freisleben and P. Merz, "A Genetic Local Search Algorithm for Solving Symmetric and Asymmetric Traveling Salesman Problems," in *Proceedings of the 1996 IEEE International Conference on Evolutionary Computation*, (Nagoya, Japan), pp. 616–621, 1996.

[12] B. Freisleben and P. Merz, "New Genetic Local Search Operators for the Traveling Salesman Problem," in *Proceedings of the 4th Conference on Parallel Problem Solving from Nature - PPSN IV*, (H.-M. Voigt, W. Ebeling, I. Rechenberg, and H.-P. Schwefel, eds.), pp. 890–900, Springer, 1996.

[13] L. M. Gambardella and M. Dorigo, "Ant-Q: A Reinforcement Learning Approach to the Traveling Salesman Problem," in *Proc. 12th International Conference on Machine Learning*, pp. 252–260, Morgan Kaufmann, 1995.

[14] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, New York, 1979.

[15] M. Gorges-Schleuter, "ASPARAGOS: An Asynchronous Parallel Genetic Optimization Strategy," in *Proceedings of the Third International Conference on Genetic Algorithms*, (J. D. Schaffer, ed.), pp. 422–427, Morgan Kaufmann, 1989.

[16] M. Gorges-Schleuter, *Genetic Algorithms and Population Structures - A Massively Parallel Algorithm*. PhD thesis, University of Dortmund, Germany, 1991.

[17] J. Grefenstette, R. Gopal, B. Rosimaita, and D. V. Gucht, "Genetic Algorithms for the Traveling Salesman Problem," in *Proceedings of an International Conference on Genetic Algorithms and their Applications*, pp. 160–168, 1985.

[18] A. Homaifar, S. Guan, and G. E. Liepins, "A New Approach to the Traveling Salesman Problem by Genetic Algorithms," in *Proceedings of the 5th International Conference on Genetic Algorithms*, pp. 460–466, Morgan Kaufmann, 1993.

[19] D. S. Johnson, "Local Optimization and the Traveling Salesman Problem," in *Proceedings of the 17th International Colloquium on Automata, Languages and Programming*, pp. 446–461, Springer, Berlin, 1990.

[20] D. S. Johnson and L. A. McGeoch, "The Traveling Salesman Problem: A Case Study in Local Optimization," in *Local Search in Combinatorial Optimization*, (E. H. L. Aarts and J. K. Lenstra, eds.), Wiley and Sons, New York, 1996. to appear.

[21] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by Simulated Annealing," *Science*, vol. 220, pp. 671–680, 1983.

[22] E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys, *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*. New York: Wiley and Sons, 1985.

[23] S. Lin and B. Kernighan, "An Effective Heuristic Algorithm for the Traveling Salesman Problem," *Operations Research*, vol. 21, pp. 498–516, 1973.

[24] K.-T. Mak and A. J. Morton, "Distances between Traveling Salesman Tours," *Discrete Applied Mathematics and Combinatorial Operations Research and Computer Science*, vol. 58, pp. 281–291, 1995.

[25] O. Martin, S. W. Otto, and E. W. Felten, "Large-Step Markov Chains for the Traveling Salesman Problem," *Complex Systems*, vol. 5, pp. 299–326, 1991.

[26] P. Merz, *Genetische Algorithmen für kombinatorische Optimierungsprobleme*. Master's thesis, University of Siegen, Germany, 1996.

[27] H. Mühlenbein, M. Gorges-Schleuter, and O. Krämer, "Evolution Algorithms in Combinatorial Optimization," *Parallel Computing*, vol. 7, pp. 65–88, 1988.

[28] H. Mühlenbein, "Evolution in Time and Space – The Parallel Genetic Algorithm," in *Foundations of Genetic Algorithms*, (G. J. E. Rawlins, ed.), pp. 316–337, Morgan Kaufmann Publishers, 1991.

[29] M. Padberg and G. Rinaldi, "Optimization of a 532-city Symmetric Traveling Salesman Problem by Branch and Cut," *Operations Research Letters*, vol. 6, pp. 1–7, 1987.

[30] J.-Y. Potvin, "The Traveling Salesman Problem: A Neural Network Perspective," *ORSA Journal on Computing*, vol. 5, pp. 328–348, 1993.

[31] G. Reinelt, "TSPLIB— A Traveling Salesman Problem Library," *ORSA Journal on Computing*, vol. 3, no. 4, pp. 376–384, 1991.

[32] G. Reinelt, *The Traveling Salesman: Computational Solutions for TSP Applications*. Vol. 840 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, Germany, 1994.

[33] J. Y. Suh and D. V. Gucht, "Incorporating Heuristic Information into Genetic Search," in *Genetic Algorithms and their Applications: Proceedings of the Second International Conference on Genetic Algorithms*, pp. 100–107, Lawrence Erlbaum, 1987.

[34] A. Y. C. Tang and K. S. Leung, "A Modified Edge Recombination Operator for the Travelling Salesman Problem," in *Parallel Problem Solving from Nature - Proceedings of the Third Workshop, PPSN III*, (H.-P. Schwefel and R. Männer, eds.), (Berlin, Germany), pp. 180–188, Springer, 1994.

[35] N. L. J. Ulder, E. H. L. Aarts, H. J. Bandelt, P. J. M. van Laarhoven, and E. Pesch, "Genetic Local Search Algorithms for the Traveling Salesman Problem," in *Parallel Problem Solving from Nature - Proceedings of 1st Workshop, PPSN I*, (H. P. Schwefel and R. Männer, eds.), (Berlin, Germany), pp. 109–116, Springer, 1991.

Appendix

I. Computational Results for the STSP

d198.tsp			
eval/gen	best	average	t in s
20/ 1	15787 (0.04 %)	15797.6 (0.11 %)	8
60/ 5	15780 (0.00 %)	15782.2 (0.01 %)	30
110/ 10	15780 (0.00 %)	15780.5 (0.00 %)	49
210/ 20	15780 (0.00 %)	15780.2 (0.00 %)	87
410/ 40	15780 (0.00 %)	15780.1 (0.00 %)	163

lin318.tsp			
eval/gen	best	average	t in s
40/ 1	42029 (0.00 %)	42244.5 (0.51 %)	5
220/ 10	42029 (0.00 %)	42083.3 (0.13 %)	39
420/ 20	42029 (0.00 %)	42061.2 (0.08 %)	68
820/ 40	42029 (0.00 %)	42040.4 (0.03 %)	118
1220/ 60	42029 (0.00 %)	42029.0 (0.00 %)	168

pcb442.tsp			
eval/gen	best	average	t in s
40/ 1	50950 (0.34 %)	51075.7 (0.59 %)	2
220/ 10	50833 (0.11 %)	50889.6 (0.22 %)	20
820/ 40	50778 (0.00 %)	50791.4 (0.03 %)	65
2020/100	50778 (0.00 %)	50780.1 (0.00 %)	135
2820/140	50778 (0.00 %)	50778.0 (0.00 %)	178

att532.tsp			
eval/gen	best	average	t in s
40/ 1	27835 (0.54 %)	27902.3 (0.78 %)	4
420/ 20	27710 (0.09 %)	27727.5 (0.15 %)	63
820/ 40	27703 (0.06 %)	27707.7 (0.08 %)	106
1620/ 80	27686 (0.00 %)	27699.4 (0.05 %)	183
2420/120	27686 (0.00 %)	27698.6 (0.05 %)	255
2820/140	27686 (0.00 %)	27698.4 (0.04 %)	290

rat783.tsp			
eval/gen	best	average	t in s
40/ 1	8876 (0.80 %)	8893.5 (0.99 %)	3
820/ 40	8815 (0.10 %)	8820.2 (0.16 %)	90
2020/100	8806 (0.00 %)	8809.4 (0.04 %)	174
4020/200	8806 (0.00 %)	8807.3 (0.01 %)	298
6020/300	8806 (0.00 %)	8806.2 (0.00 %)	424

fl1577.tsp			
eval/gen	best	average	t in s
40/ 1	22362 (0.71 %)	22527.2 (1.46 %)	115
820/ 40	22257 (0.24 %)	22270.0 (0.30 %)	2582
2020/100	22250 (0.21 %)	22254.7 (0.23 %)	5134
4020/200	22249 (0.20 %)	22251.0 (0.21 %)	8981
6020/300	22249 (0.20 %)	22250.5 (0.21 %)	12762

fl3795.tsp			
eval/gen	best	average	t in s
40/ 1	30102 (4.80 %)	30691.6 (6.85 %)	184
820/ 40	28872 (0.52 %)	29237.0 (1.79 %)	3503
2020/100	28795 (0.25 %)	28971.0 (0.86 %)	6404
4020/200	28783 (0.21 %)	28893.8 (0.59 %)	10259
6020/300	28778 (0.19 %)	28878.7 (0.54 %)	14097
8020/400	28776 (0.18 %)	28868.5 (0.51 %)	17824

Fig. 4. The Results for Symmetric TSP Instances

II. Computational Results for the ATSP

p43.atsp			
step/gen	best	average	t in s
80/ 1	5621 (0.02 %)	5622.6 (0.05 %)	< 1
240/ 5	5620 (0.00 %)	5621.2 (0.02 %)	1
440/ 10	5620 (0.00 %)	5620.9 (0.02 %)	2
840/ 20	5620 (0.00 %)	5620.5 (0.01 %)	4
2440/ 60	5620 (0.00 %)	5620.1 (0.00 %)	11

ry48p.atsp			
step/gen	best	average	t in s
80/ 1	14765 (2.38 %)	14901.2 (3.32 %)	1
1640/ 40	14422 (0.00 %)	14525.8 (0.72 %)	5
8040/200	14422 (0.00 %)	14479.5 (0.40 %)	21
16040/400	14422 (0.00 %)	14462.0 (0.28 %)	41
28040/700	14422 (0.00 %)	14451.2 (0.20 %)	72

ft70.atsp			
step/gen	best	average	t in s
80/ 1	38782 (0.28 %)	39072.9 (1.03 %)	4
240/ 5	38736 (0.16 %)	38842.7 (0.44 %)	23
440/ 10	38677 (0.01 %)	38737.6 (0.17 %)	41
840/ 20	38675 (0.01 %)	38680.5 (0.02 %)	68
1640/ 40	38673 (0.00 %)	38674.2 (0.00 %)	111

kro124p.atsp			
step/gen	best	average	t in s
80/ 1	38281 (5.66 %)	38616.6 (6.59 %)	1
1640/ 40	36326 (0.27 %)	36487.5 (0.71 %)	11
4040/100	36230 (0.00 %)	36264.8 (0.10 %)	21
7240/180	36230 (0.00 %)	36232.5 (0.01 %)	33
8040/200	36230 (0.00 %)	36231.5 (0.00 %)	35

ftv170.atsp			
step/gen	best	average	t in s
80/ 1	2866 (4.03 %)	2936.1 (6.57 %)	1
840/ 20	2800 (1.63 %)	2818.7 (2.31 %)	7
1640/ 40	2767 (0.44 %)	2780.7 (0.93 %)	12
4040/100	2755 (0.00 %)	2764.3 (0.34 %)	27
16040/400	2755 (0.00 %)	2762.2 (0.26 %)	100

Fig. 5. The Results for Asymmetric TSP Instances