Interactive Genetic Algorithms for the Traveling Salesman Problem

Sushil J. Louis
Genetic Adaptive Systems Lab
Dept. of Computer Science/171
University of Nevada, Reno
Reno, NV 89557
sushil@cs.unr.edu

Rilun Tang
Genetic Adaptive Systems Lab
Dept. of Computer Science/171
University of Nevada, Reno
Reno, NV 89557
tang@cs.unr.edu

Abstract

We use an interactive genetic algorithm to divide and conquer large traveling salesperson problems. Current genetic algorithm approaches are computationally intensive and may not produce acceptable tours within the time available. Instead of applying a genetic algorithm to the entire problem, we let the user interactively decompose a problem into subproblems, let the genetic algorithm separately solve these subproblems and then interactively connect subproblem solutions to get a global tour for the original problem. Our approach significantly reduces the computing time to find high quality solutions for large traveling salesperson problems. We believe that an interactive approach can be extended to other visually decomposable problems.

1 INTRODUCTION

The traveling salesperson problem (TSP) is a classical example of an NP-Hard combinatorial optimization problem (Garey and Johnson, 1979). Given N cities and distances among them, the aim is to find the shortest tour that visits each city once and ends at the city it started from. Researchers have tried various algorithms to solve this problem with the aim usually being just to find near optimal solutions. The algorithms include simulated annealing (Learhoven and Aarts, 1987) (Kirkpatrick and Toulouse, 1985), discrete linear programming (Crowder and Padberg, 1980), neural networks (Aarts and Stehouwer, 1993), branch and bound (Padberg and Rinaldi, 1987), 2-opt (Lin and Kernighan, 1973), Markov chain (Martin et al., 1991) and genetic algorithms.

Genetic Algorithms (GAs), first developed by Holland in the 1970s (Holland, 1975), are search algorithms based on the mechanics of natural selection and natural genetics. They are used to search large, non-linear search spaces where expert knowledge is lacking or difficult to encode and where traditional optimization techniques fall short (Goldberg and Lingle, 1985; Davis, 1985; Louis, 1993). Since the 80's, much work has been done in applying genetic algorithms to the TSP (Goldberg and Lingle, 1985; Grefenstette et al., 1985; Oliver et al., 1987; Jog et al., 1989; Whitley et al., 1989). This literature and more recent work has resulted in a plethora of GA operators and techniques for attacking large TSPs (Starkweather et al., 1991; Whitley et al., 1991; Hamaifar et al., 1993; Schmitt and Amini, 1998; Jog et al., 1991) and has produced significant improvements in this area. GAs are suitable for the problem because they quickly direct search to promising areas of the search space, but, there is a paucity of approaches capable of both high solution quality and speed. If we use TSP problem size, selection scheme (generational or steady state), population initialization, population size and crossover function as experimental factors, none of these factors has significant impact on the CPU time when large TSPs were solved (Schmitt and Amini, 1998). In addition, GAs are quite slow when implemented on a sequential machine (Jog et al., 1991). Valenzuela used a new GA approach, evolutionary divide and conquer (EDAC), trying to solve the scaling and the solution quality degradation problems for large TSPs (Valenzuela, 1995). EDAC produces good results on tour length, but the running time is still quite long.

In this paper, we propose a new methodology, for applying genetic algorithms to solve large TSPs. The Interactive Genetic Algorithm (IGA) asks the user to divide the original large number of cities into smaller clusters each of which contains fewer cities. A GA then solves these smaller TSPs. After finding the optimal or near-optimal solution for each cluster, the user helps connect pairs of clusters until a global tour emerges. The visualization and interaction interface is written in the Java language. We are interested in exploring the use of interaction in genetic search and started with TSPs because they are so easily visualizable.

Interactive evolution (IE) (Banzhaf, 1997) is another evolutionary algorithm that needs human interaction. In IE, the user selects one or more individual(s) which survive(s) and reproduce(s) (with variation) to constitute a new generation. Our IGA is very different from IE in that the IGA uses interaction for problem formulation or solution repair. In contrast, interactive evolution uses user interaction to obtain fitness information and the user assumes an active role in the search process.

We chose to use greedy crossover (Grefenstette et al., 1985), CHC selection (Eshelman, 1991) and a new greedy-swap mutation operator in our genetic algorithm. Our results indicate that the interactive genetic algorithm quickly provides quality results on large TSPs. Specifically, the running time is much smaller for the IGA compared with the running time of a GA on the same problem. The choice of a particular genetic algorithm is of less significance for this comparative result and you may substitute other evolutionary computing algorithms. We believe that IGAs will be especially useful on the many large TSPs with structured, as opposed to randomized, distributions of cities.

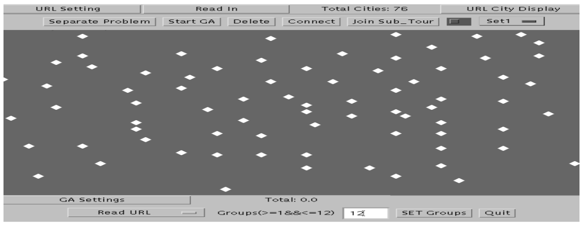
The next section describes our setup and methodology. Section 3 provides some results and analysis. The last section furnishes conclusions and directions for future work.

2 METHODOLOGY

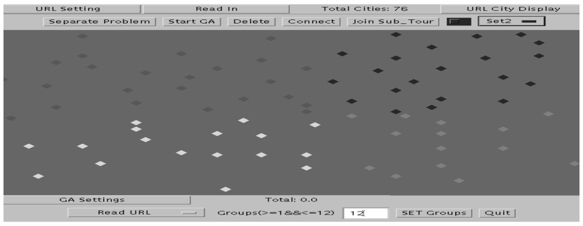
Using the IGA to attack TSPs follows four steps.

Step 1: Plot and display city locations on the interface as shown in Figure 1 (a). This provides the user with ability to identify and exploit any structure in the distribution of cities. Although we could use one of several clustering methods to decompose the problem, humans are very good at finding patterns in visual data – we simply exploit this property.

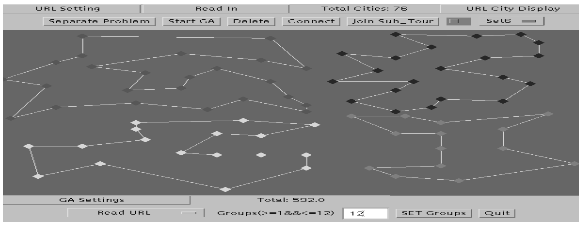
Step 2: Separate the data into clusters Figure 1 (b). The clusters are separated manually by choosing a cluster id and then using the mouse to select groups of points (cities) on the interface to belong to the chosen cluster. The current version allows all cities inside a mouse drawn rectangle to be assigned to a cluster.



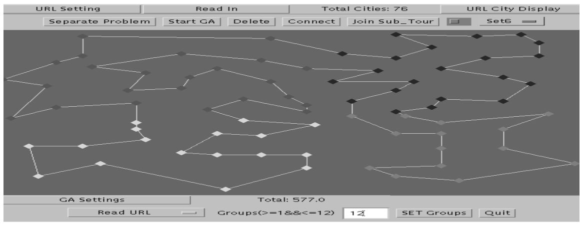
(a) Display city coords



(b) User decomposes the TSP



(c) Run GA on each cluster to get sub-tours



(d) Connect sub-tours to get a global tour

Figure 1: IGA Procedure (http://gaslab.cs.unr.edu/)

Cities can also be added and deleted from clusters.

Step 3: Run the genetic algorithm on each cluster. Note that this can be done in parallel with one GA assigned to each cluster. We can run each GA on a different machine or do further parallelization. Our Java implementation creates a separate thread for each cluster while the C implementation simply starts separate processes. At the end of this step we get tours for each cluster of cities as shown in Figure 1 (c).

Step 4: Connect these sub-tours to get a complete tour. We currently allow two ways of reconnecting sub-tours.

1. Manual: The user can choose to delete and add edges to open subtours and to combine adjacent open subtours. The current sum of the lengths of all edges (total tour length) is always displayed so that the user can experiment with the subtours. In this scheme, the whole process is based completely on the user's visual judgment. That is, we try to decrease the total tour length by replacing long edges with short edges.

2. Semi-Automatic: The user draws a rectangle that includes appropriate (in the user's judgment) cities in two adjacent clusters and the interface does an exhaustive search among chosen city edges to find edges to delete and add such that the combined tour length is minimized. The total tour length will also be updated. This scheme relieves the user from the tedium of experimenting with fine adjustments – you only need to specify a promising area for reconnection.

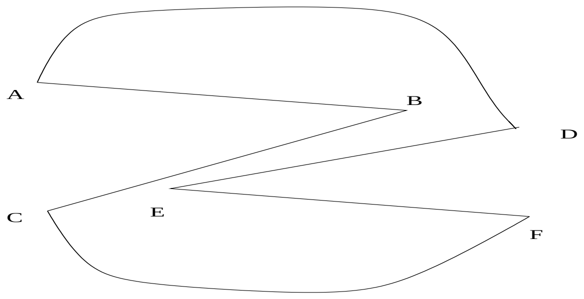
2.1 Genetic Algorithm for TSP

Our encoding and operators for the genetic algorithm itself are not particularly novel and we describe them below. The objective function for the N cities two dimensional Euclidean TSP is the sum of Euclidean distances between every pair of cities in the tour. That is:

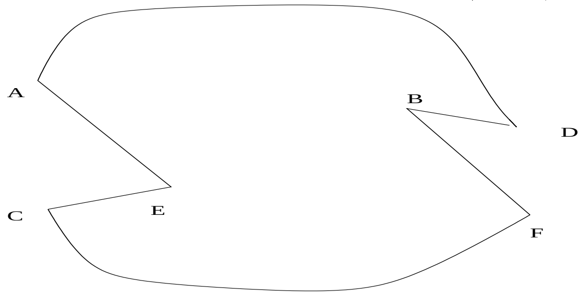
$$\text{Obj. function} = \sum_{i=1}^{N} \sqrt{(x_i - x_{i-1})^2 + (y_i - y_{i-1})^2}$$

Where, x_i, y_i are the coordinates of city i and x_N, y_N equals x_0, y_0. We turn this minimization problem into a maximization problem for the GA by subtracting the objective function value from a large constant.

We use the usual path representation where the cities are listed in the order in which they are visited. Greedy crossover which was invented by Grefenstette



A tour before applying greedy-swap (Tour1)



A possible tour after greedy-swap (Tour2)

Figure 2: Illustration of the Greedy-Swap Mutation

(Grefenstette et al., 1985) ensures that we always get legal tours.

We use a new mutation operator, greedy–swap. The basic idea of greedy–swap is to randomly select two cities from one chromosome and swap them if the new (swapped) tour length is shorter than the old one. Only four edges are different between the two tours. This is shown in Figure 2. Before mutation we have $Tour1 = \underbrace{DA}B\underbrace{CF}ED$, and city B and city E are randomly chosen from the tour as the swap positions. After swapping B and E, we get tour $Tour2 = \underbrace{DA}E\underbrace{CF}BD$. Edges AB, BC, FE and ED in Tour1 are replaced by edges AE, EC, FB and BD in Tour2. We keep the new tour only when

$$\begin{aligned}&\|AB\| + \|BC\| + \|FE\| + \|ED\| \\ &> \|AE\| + \|EC\| + \|FB\| + \|EB\|\end{aligned}$$

All the other edges are the same in Tour1 and Tour2, so the total tour length of Tour2 when the above condition holds, is shorter.

We use CHC selection to guarantee that the best individual will always survive in the next generation (Eshelman, 1991). In CHC selection if the population size is N, we generate N children by roulette wheel selection, then combine the N parents with the N children, sort these 2N individuals according to their fitness value and choose the best N individuals to propagate to the next generation.

2.2 Analysis

The compelling advantage of our approach follows from the fact that the sum of factorials is much smaller than the factorial of the sum. For a TSP with N cities, the search space is a function of $N!$. Thus the computing time, which is proportional to the search space, is also a function of $N!$. When N is a large number, the computing time is extremely long.

When decomposing the problem, if N cities are divided into P clusters, the average number of cities in each cluster in N/P. Therefore, the search space for each cluster is a function of $(N/P)!$. The total search space for all clusters is: $P(N/P)!$. When N is large, our approach will save several orders of magnitude worth of time since

$$P \times \left(\frac{N}{P}\right)! \ll N!$$

for large N and P.

3 RESULTS AND ANALYSIS

We applied the IGA to eight (8) different TSPs (eil51, eil76, ch150, bier127, a280, d657, lin318, vm1084) from the TSPLIB (Reinelt, 1996) that span a wide range of sizes. These TSPs have been used by others and are considered benchmarks with optimal or best known tour lengths available. We follow the steps outlined in section 2 for each problem and compare our results with an identical GA running on the complete problem. For every problem, we ensure that both the IGA and the GA complete the same number of evaluations. For the IGA the number of evaluations is given by

$$\text{evaluations} = P \times IGA_g \times IGA_p$$

where P is the number of clusters, IGA_g is the number of generations run, and IGA_p is the populations size (identical for each cluster). Since we tried to decompose the problem into equal sized clusters, we used the same population size across all clusters on a particular TSP instance. The number of evaluations for the GA on the complete TSP is $GA_g \times GA_p$ and

$$P \times IGA_g \times IGA_p = GA_g \times GA_p$$

for all problems. We ran all problems on the same unloaded ULTRA5 Sun workstations. The GA code used by the IGA was the same as the code run on complete problems.

Table 1 compares the performance of the IGA with the GA. Column one shows the size of the problem, the next three columns deal with the IGA and the last two columns provide results from the GA. Column two lists the number of clusters used. Since we found that the GA is able to competently solve TSPs with less than one hundred cities, we chose a number of clusters that would result in subproblems with less than one hundred cities. Column three shows the time in minutes from the start of interaction until obtaining a complete tour with the IGA. The difference in time is more than clear at the wristwatch level of resolution. Column four lists the quality of solution in percentage over optimal. The last two columns provide time in minutes and the quality of solution obtained by the GA.

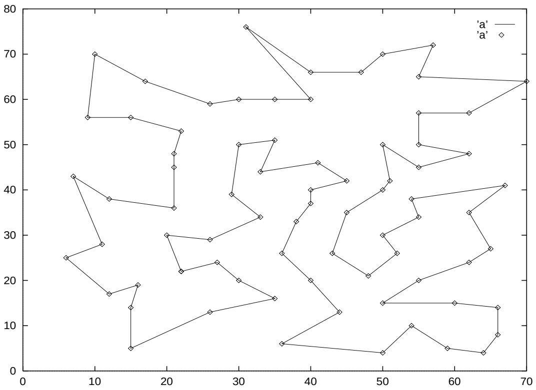
Table 1: Results from the IGA and the GA for different size TSPs

No. of Cities	No. of Sets	IGA 1 run Time (min)	IGA 1 run % Over optimal	GA 10 runs Time (min)	GA 10 runs % over optimal
51	2	0.25	0.7	0.96	7.0
76	2	0.60	5.9	2.95	13.0
127	2	2.70	11.7	16	13.1
150	5	2.70	9.1	26.4	13.1
280	7	6.16	12.2	145.9	22.0
318	7	5.45	14.2	181.6	33.9
657	12	11.55	32.2	296.2	44.0
1084	12	40.1	26.0	684.5	72.0

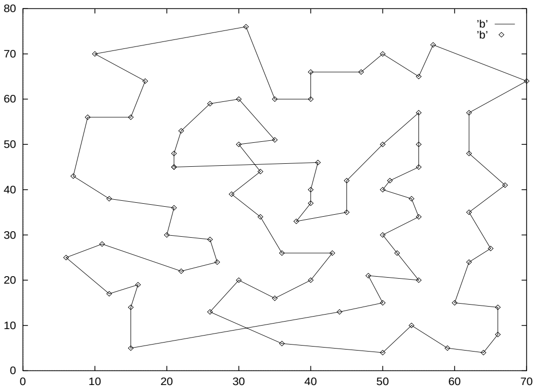
If N is the number of cities, we let *small* TSPs be categorized as those with $N \leq 100$, *medium* TSPs are those with $(100 < N \leq 200)$ and *large* TSPs have $N > 200$. From the table, we can see that the IGA easily outperforms the GA on large problems.

3.0.1 Small TSPs

For small TSPs (eil51, eil76), both methods produce similar results. The IGA gets good results (within 10% of the benchmarks) in less than one minute, so does the GA. Here we only tried to separate each problem into two sub-groups and the running time is the same for both methods. Fig 3 show the best tours obtained by both approaches on the 76 city problem. Note that we got a tour without crossings with the IGA.

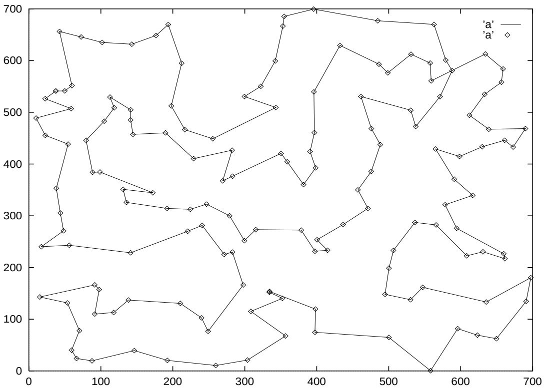


(a) Best tour using the IGA

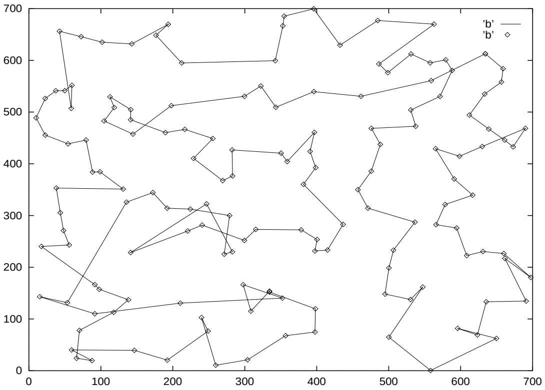


(b) Best Tour using the GA

Figure 3: 76 city problem



(a) Best tour using the IGA



(b) Best tour using the GA

Figure 4: 150 city problem

3.0.2 Medium TSPs

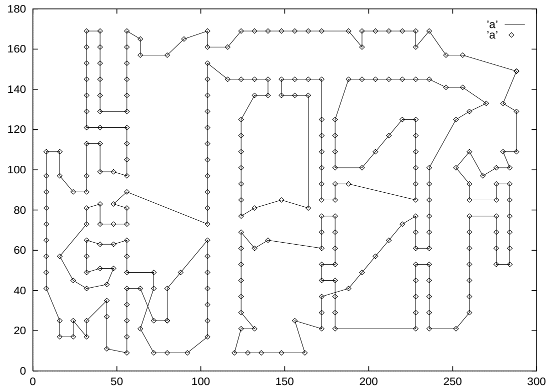
We tried two problems – the 127 city problem and the 150 city problem. We can start seeing the difference in time and quality at this stage. The IGA does better, both in running time and tour length.Fig 4 shows the best tours on the 150 city problem. There is a marked increase in the number of crossings for the tour produced by the GA when compared to the IGA tour. Since the number of crossings correlates with tour length the GA's tour is longer than IGA's tour. Results from the 127 city problem are similar.

3.0.3 Large TSPs

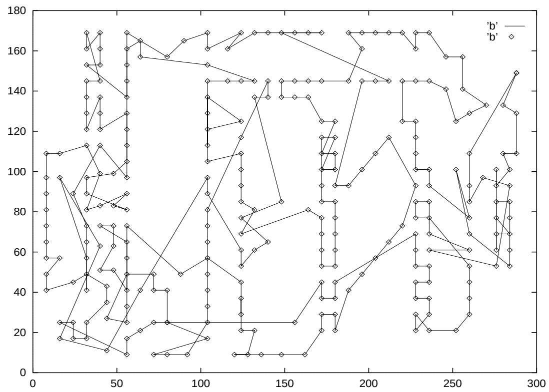
Four problems can be classified as large TSPs – 280 city, 318 city, 657 city and 1084 city. The running time with the IGA is now significantly reduced compared to the GA with much better tours being produced.

We separated the 280 city and 318 city problems into 4-6 clusters for the IGA. The best tours for 280 city problem are shown in Figure 5. Such large structured problems may be most suited to our approach. We can get a good tour in minutes with the IGA (Figure 5 (a)) while even after two hours we can only get the tour shown in Figure 5 (b) for the genetic algorithm. Note also that we can visually repair complete tours produced by the GA or the IGA. For example, we could remove the two crossings in Figure 5 (a) by deleting and adding edges during post processing. There is, of course, a limit to this kind of repair.

On really large problems, like the 657 city or the 1084 city problems, the GA needs to run for hours. After separating these problems into about 12 clusters the IGA gets good tours within an hour. Fig 6 (a) show the best tour obtained by the IGA on the 1084 city problem. Compare this tour with the messy tour produced by the GA on the same problem.

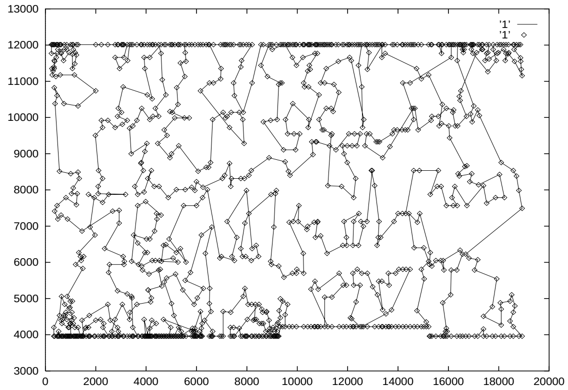


(a) Best tour using the IGA

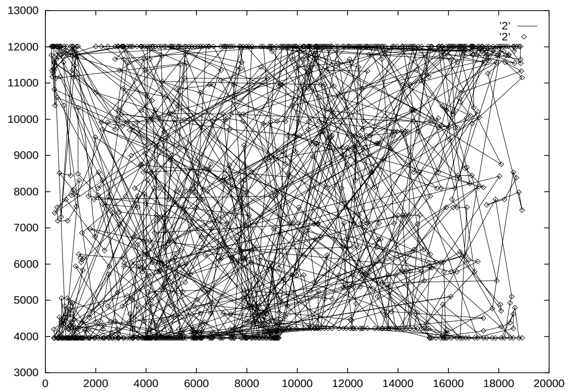


(b) Best tour using the GA

Figure 5: 280 city problem



(a) Best tour using the IGA



(b) Best tour using the GA

Figure 6: 1084 city problem

3.1 Analysis

3.1.1 Number of clusters

For a given number of cities and approximately equal clusters, the larger the number of clusters P, the faster the IGA (Section 2.2). On the other hand, larger P results in a more local view of optimization with a large number of (potentially critical) tours not being explored. From our experiments, genetic algorithms do well on small TSPs with up to 100 cities. Therefore, when decomposing, it is appropriate to keep each cluster within $60 - 100$ cities.

3.1.2 Clustering

Even when we run the IGA on the same TSP with the same number of clusters, we can get different results because we may not choose exactly the same set of cities for each cluster. Currently, the user manually decomposes the problem and uses the visualization provided by the interface to aid this clustering process. Not only are no two users exactly alike, the same user may not provide the same classification. Without completely getting rid of all interaction we plan to use available clustering techniques to provide an initial clustering that the user can then modify. The automatic clustering will get the same clusters on the same problem each time we run it, the user can then visually adjust these clusters.

4 CONCLUSIONS

We presented a new interactive genetic algorithm to solve the traveling salesperson problem. A user visually partitioned the TSP into sub-problems and the GA solved each subproblem separately. The user then visually recombined the sub-tours into a global tour. We tested this methodology with a set of TSPs benchmarks using a relatively untuned genetic algorithm and compared the results with the same GA running on the complete un-decomposed problem. The IGA takes much less time to provide better quality results for large problems. For TSPs with more than 200 cities, this methodology greatly reduced the running time.

Although we can divide and conquer problems where the cities are randomly distributed, non-random distributions amenable to clustering bring out the real strength of our approach. Clusters can be easily picked out and the interactive genetic algorithm can exploit this structure to quickly solve the problem. We believe that this method can be extended to other visualizable, decomposable problems. A Java version of the IGA can be found and used from http://gaslab.cs.unr.edu/.

We did not expend much effort in tuning our genetic algorithm for the TSP. Using a well-tuned GA or other search algorithm that quickly and near-optimally solves small TSPs (from our decomposition) should also result in better quality global tours (Jog et al., 1989). We plan to investigate other interactive genetic algorithm applications.

Acknowledgments

This material is based upon work supported by the National Science Foundation under Grant No. 9624130.

References

Aarts, E. H. L. and Stehouwer, H. P. (1993). Neural networks and the traveling salesman problem. In *Proc. Int. Conf. on Artifical Neural Networks*. Spring Verlog.

Banzhaf, W. (1997). *Interactive Evolution*. IOP Publishing Ltd and Oxford University Press.

Crowder, H. and Padberg, M. W. (1980). Solving large scale symmetric traveling salesman problems to optimality. In *Management Science, 26:495-509*.

Davis, L. (1985). Job shop scheduling with genetic algorithms. In *Proceedings of the Second International Conference on Genetic Algorithms*. Lawrence Eribaum Associates, Mahwah, NJ.

Eshelman, L. J. (1991). *The CHC Adaptive Search Algorithm: How to Have Safe Search When Engaging in Nontraditional Genetic Recombination*. Morgan Kauffman.

Garey, M. R. and Johnson, D. S. (1979). *Computers and interactability: a guide to the theory of NP-completeness*. Freeman San Francisco.

Goldberg, D. and Lingle, R. (1985). Alleles, loci and the traveling salesman problem. In *Proceedings of the Second International Conference on Genetic Algorithms*, Mahwah, NJ. Lawrence Eribaum Associate.

Grefenstette, J., Gopal, R., Rosmaita, R., and Gucht., D. (1985). Genetic algorithms for the traveling salesman problem. In *In Proceedings of the Second International Conference on Genetic Algorithms*. Lawrence Eribaum Associates, Mahwah, NJ.

Hamaifar, L., Guan, C., and Liepins, G. (1993). A new approach to the traveling salesman problem by genetic algorithms. In *Proceedings of the Fifth International Conference on Genetic Algorithms*. Los Altos, CA:Morgan Kaufmann Pulbishers.

Holland, J. (1975). *Adaptation In Natural and Artificial Systems*. The University of Michigan Press, Ann Arbor.

Jog, P., Suh, J. Y., and Gucht, D. V. (1989). The effect of population size, heuristic crossover and local improvement on a genetic algorithm for the traveling salesman problem. In *Proceedings of the Third International Conference on Genetic Algorithms*. Los Altos, CA:Morgan Kaufmann Pulbishers.

Jog, P., Suh, J. Y., and Gucht, D. V. (1991). Parallel genetic algorithms applied to the traveling salesman problem. In *SIAM J. Optimization 1:515-529*.

Kirkpatrick, S. and Toulouse, G. (1985). Configuration space analysis of travelling salesman problems. In *Journal de Physiqu 46(8):1277-1292*.

Learhoven, P. V. and Aarts, E. H. L. (1987). *Simulated Annealing:The Theory and Application*. Kluwer Academic Publishers.

Lin, S. and Kernighan, B. (1973). An effective heuristic algorithm for the travelling-salesman problem. In *Operations Research, 21(2):498-516*.

Louis, S. J. (1993). Genetic algorithms as a computational tool for design. In *PhD thesis*. Indiana University, Indiana University.

Martin, O., Otto, S., and Felten, E. (1991). Large-step markov chains for the traveling salesman problem. In *Complex Systems, 5(3):299-326*.

Oliver, I. M., Smith, D. J., and Holland, J. R. (1987). A study of permutation crossover operators on the traveling salesman prolbem. In *Proceedings of the Third International Conference on Genetic Algorithms*. London: Lawrence Eribaum Associates.

Padberg, M. and Rinaldi (1987). Optimization of a 532-city symmetric traveling salesman problem by branch and cut. In *Operations Research Letters 6(1):1-7*.

Reinelt, G. (1996). http://www.iwr.uni-heidelberg.de/iwr/comopt/soft/tsplib95/tsplib.html.

Schmitt, L. J. and Amini, M. N. (1998). Performance characteristics of alternative genetic algorithmic approaches to the traveling salesman problem using path representation: An empirical study. In *European Journal of Operational Research 108: 551-570.*

Starkweather, T., Whitley, D., Whitley, C., and Mathial, K. (1991). A comparison of genetic sequencing operators. In *Proceedings of the Fourth International Conference on Genetic Algorithms*. Los Altos, CA:Morgan Kaufmann Pulbishers.

Valenzuela, C. L. (1995). *Evolutionary Divide and Conquer: a novel genetic approach to th e TSP.* PhD thesis, Imperial College, University of London, London, England.

Whitley, C., Starkweather, T., and Shaner, D. (1991). The traveling salesman and sequence scheduling quality solutions using genetic edge recombination. In *Handbook of Genetic Algorithms*. New York:Van Nostrand Reinhold.

Whitley, D., Starkweather, T., and Fuquay, D. (1989). Scheduling problems and traveling salesman: The genetic edge recombination operator. In *Proceedings of the Third International Conference on Genetic Algorithms*. Los Altos, CA:Morgan Kaufmann Pulbishers.