Proceedings of the 5th WSEAS International Conference on Applied Computer Science, Hangzhou, China, April 16-18, 2006 (pp101-108)

Improvement heuristics for the Sparse Travelling Salesman Problem

FREDRICK MTENZI
Computer Science Department
Dublin Institute of Technology
School of Computing, DIT Kevin Street, Dublin 8
IRELAND
http://www.comp.dit.ie/fmtenzi

Abstract: - The Sparse Travelling Salesman Problem (Sparse TSP) is a variant of the Travelling Salesman Problem (TSP), which is one of the major success stories in optimization. The TSP can be described as the problem of finding a route of a salesman starting from his home city, visiting each city in a particular region exactly once and returning home at the end while minimizing the tour length. The Sparse TSP which is studied in this paper is a problem of finding the shortest route of the salesman when visiting cities in a region making sure that each city is visited at least once and returning home at the end. In the Sparse TSP, the distance between cities may not obey the triangle inequality (i.e., the shortest distance between any two cities may not be a direct road joining the two cities; it may be cheaper to go via other cities). In this paper we design and implement improved versions of *2-opt* and *3-opt* heuristic algorithms, which are specifically designed to take advantage of sparsity in the Sparse TSP. These improvement heuristic algorithms incorporate the use of large neighbourhood structure, enabling them to produce results which are much better than existing ones. In our implementation we use several, speed-up techniques to make our algorithms run faster. We test our improvement heuristic algorithms using problems taken from road network in rural Ireland and the TSP Library (TSPLIB).

Key-Words: - Sparse TSP, heuristics, neighbourhood structure, optimization, tour, local search

1 Introduction

The Sparse Travelling Salesman Problem (Sparse TSP) is a variant of the Travelling Salesman Problem (TSP), which is one of the major success stories in optimization. There are many real world problems which may be formulated as instances of the TSP such as very large scale integration (VLSI) chip manufacturing and drilling printed circuit boards. The TSP can be described as the problem of finding a route of a salesman starting from his home city, visiting each city in a particular region exactly once and returning home at end while minimizing the tour length. The Sparse TSP which is studied in this paper is a problem of finding the shortest route of the salesman when visiting cities in a region making sure that each city is visited at least once and returning home at the end. In the Sparse TSP, the distance between cities may not obey the triangle inequality (i.e., the shortest distance between any two cities may not be a direct road joining the two cities; it may be cheaper to go via other cities). An intriguing aspect of the problem is the relative ease with which it can be described and the extreme difficulty it presents in finding the optimal solution. This problem has been shown to be NP-hard [1] and therefore finding optimal solution to it is extremely difficult.

One approach of finding solutions to the Sparse TSP problem is by using improvement heuristic algorithms. The main idea is to improve a feasible solution by performing a series of transformations (alterations) or moves. The transformations are normally specified by a neighbourhood function that defines which solutions can be generated by a single move. Given a feasible initial solution, the algorithm searches its neighbourhood for a better solution.

In the case of the Sparse TSP, we define the neighbourhood of a tour say, *T*, to be all those tours which can be obtained by changing at most *k* arcs of *T*. A tour is said to be locally optimal if no tour in its neighbourhood is shorter than it. We can search for local *k-opt* tours by starting with a non-optimal tour T_1 and constructing a sequence of tours $T_1, T_2, \cdots, T_m$. Each tour is obtained from the previous one by performing a *k*-change, i.e. by deleting *k* arcs and reconnecting the loose ends using *k* arcs which are not in the present tour so as to still have a tour. The *k*-change is required to decrease the length of the tour, until no more improvement can be made.

There are two ways in which the neighbourhood is searched in improvement heuristic algorithms. First improvement, in which the current solution is replaced with the first better solution found in the neighbourhood; and the best improvement, in which the current solution is replaced with the best solution in the neighbourhood. In this paper, we use the first improvement for the majority of testing.

By definition, any type of improvement heuristic terminates in a local optimum. Furthermore, the quality of the final solution as compared with the global optimum depends on the size and the structure of the neighbourhood. The time needed to verify local optimality is then proportional to the size of the neighbourhood and the time required to evaluate the objective value of any solution in the neighbourhood. Tour improvement heuristic searches a neighbourhood of polynomial size to find moves that transform the current solution into a new one.

A well-known approach for improving the solution quality of improvement heuristics is to adopt a multi-start approach in which several independent runs, each using a different starting solution, are performed, and then the best of the resulting solutions is selected. Typically, the starting solutions can be chosen randomly, or by applying some tour construction heuristic. These starting solutions do not rely on the results of previous runs of the tour construction and/or improvement heuristics. We refer to this approach as the repeated local search.

However, a far more effective approach is to allow dependent runs by generating the new starting solution from one of the previous local optima by a suitable perturbation method. Such an approach is known as iterated local search, and is a widely recognized method for obtaining high quality solutions at relatively low computational cost, without resorting to more intricate tour improvement heuristics [13].

Our computational results are obtained for the data from the road network in rural Ireland. We consider problem instances of size between 10 and 4923 cities. For each instance, we run each improvement heuristic with different starting solutions. We have done testing using data from TSPLIB which have been used in a lot of studies in the literature. Improvement heuristics designed and implemented in this study produce better results than those reported in literature. However, for random generated problems our algorithms are marginally worse.

The rest of the paper is organized as follows. The Local search and neighbourhood structure background information is discussed in section 2. In section 3 we give a detailed explanation of the repeated local search heuristics, specifically the *2-opt* and *3-opt*. Iterated local search heuristics techniques and other approaches to tour improvements are also discussed in section 3. We present a summary of our computational results and discuss these results in section 4. Finally our summary and conclusions are given in section 5.

2 Background

2.1 Local search

The most common tour improvement heuristics are the *2-opt* (see Croes [3]) and the *3-opt* algorithm proposed by Lin [4]. Here, *2* or *3* arcs are removed from the tour, all possible reinsertions are attempted and the best is implemented. These operations are repeated until no further improvement is possible. The complexity of *2-opt* and *3-opt* heuristic algorithms are $O(n^2)$ and $O(n^3)$ respectively. Lin and Kernighan [5] proposed an improvement to these algorithms. The value of *k*, (i.e. number of arcs to be deleted) is modified dynamically throughout the algorithm, but this procedure is more difficult to code than the original Lin *3-opt* method. In a similar vein, Stewart [6] described an accelerated *3-opt* version which considered neighbour tours constructed using only arcs of the *k* shortest spanning trees of *G*. Reinelt [7] and Bentley [8] have proposed other types of composite heuristics that emphasize low execution times rather than solution quality.

2.2 Neighbourhood Structure

We define neighbourhood structure as follows. Let S_n denote the set of feasible solutions associated with the TSP problem. For every solution $s \in S_n$, a subset or neighbourhood of S_n, $N(S)$, is defined. When such a neighbourhood has been defined for each $s \in S_n$, we say that a neighbourhood structure N has been defined on S_n. Given a specific parameter, *x*, a sequence of solutions in S_n is then generated as follows.

1. s_1, the initial solution.

2. s_{i+1} can be any point in $N(s_i)$ such that $c(s_{i+1}, x) < c(s_i, x)$

When for some *k*, $c(s_k, x) \le c(s, x) \forall s \in N(s_k)$, s_k is said to be locally optimal with respect to the neighbourhood structure *N*. It should be noted that s_k may not necessarily be globally optimal, but the cost elements of the sequence are strictly decreasing.

Neighbourhoods can be complex, asymmetric, and cost dependent. Three types of neighbourhoods that have been applied to the Sparse TSP include the, *k*-change neighbourhoods (Johnson and McGeoch [13]). Another by Rego [9] uses ejection chains to generate compound neighbourhood structures for the TSP. Thirdly, Gutin [10] uses exponential neighbourhoods to obtain in polynomial time the best among a very large number of tours.

A critical issue in the design of a tour improvement heuristic is the choice of the neighbourhood structure, that is, the manner in which the neighbourhood is defined. In this paper we use large neighbourhoods (k-change and a variation of ejection chains), which produce near-optimal solutions of better quality. Large neighbourhoods takes longer to search, therefore we have included a number of speed-up techniques.

3 Repeated and iterated local search

3.1 The Two-opt algorithm

A 2-opt algorithm consists of deleting two arcs from the tour and reconnecting the two resulting paths in a different way to obtain a new tour which is shorter. The arcs which are used to reconnect the new tour should not be in any of the previous tours.

The *2-opt* neighbourhood of the solution, contains all the tours that are obtained by selecting a pair of distinct, non-adjacent arcs, say (i_1, i_2) and (j_1, j_2), and replacing them by the arcs (i_1, j_1) and (i_2, j_2), as illustrated in figure 1. This is the only arc exchange that creates a feasible tour. In the *2-opt* algorithm, the new tour is accepted as the current tour if and only if it is shorter than the current tour. The size of the *2-opt* neighbourhood is $O(n^2)$.

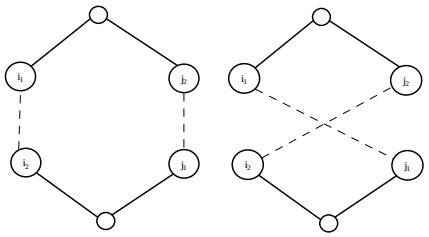

Fig. 1: A 2-opt exchange: original tour on the Left and improved tour on the right

We redesigned and implemented the *2-opt* improvement heuristic algorithm to take advantage of sparsity and utilize other large neighbourhoods. For any pair of arcs which are to be used for exchange, we checked if they existed. If they did not exist we found the shortest path between their end nodes. The modified *2-opt* algorithm with shortest path is given as follows:

The 2-opt algorithm using Existing path, shortest path and enlarged neighbourhood

Construct the initial tour *T* with length *f(T)*
Tour = f(T); Improve = 0
Repeat
If there exists any two unvisited nodes **then**
Mark them visited
Add them to the tour
else
Find a shortest path to the next unvisited pair of nodes
For *k=1* to n-2 **do**
If ExistArc $((i_k, i_{k+1}) and (j_k, j_{k+1}))$ **then**
Exchange arcs
Improve= $d(i_k, i_{k+1}) + d(j_k, j_{k+1}) - \{d(i_k, j_k) + d(i_{k+1}, j_{k+1})\}$
Else
Find a shortest path to the next unvisited pair of nodes
Exchange arcs
Improve= $d(i_k, i_{k+1}) + d(j_k, j_{k+1}) - \{P(i_k, j_k) + P(i_{k+1}, j_{k+1})\}$
EndElse
EndIf
$f(T') = f(T) - \text{Improve}$
Endfor
Until Improve=0 and all nodes are visited
$Tour = f(T')$

3.2 The Three-opt algorithm

A *3-opt* algorithm is expensive, since each move replaces two or three arcs in the current tour by two or three others not in the tour. Suppose that arcs $(i_1, i_2), (j_1, j_2)$ and (k_1, k_2) are to be replaced. If all three arcs are to be replaced and they are non-adjacent, then there are four triples of new arcs that will produce a tour. If all three arcs are to be

replaced and exactly two of them are adjacent, then *3-opt* becomes the node reinsertion algorithm explained in Reinelt [11] and the new arcs are uniquely determined. Also, each pair of non-adjacent arcs can be replaced to give a *2-opt* tour. Since a tour has $O(n^3)$ neighbours under a *3-opt* search, verifying local optimality takes $O(n^3)$ time.

In the *3-opt* exchange, removal of three arcs from the current tour results in three paths which can be joined with three arcs not in the tour to form a new tour in eight possible ways. Since we are assuming that the tour is already *2-opt*, only two cases needs to be considered in the *3-opt* case. These cases are shown in figure 2.

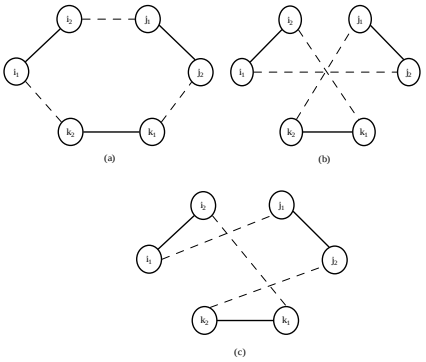

Fig. 2: Possible 3-opt exchanges: (a) is the Original Tour, (b) and (c) are improved tours

The *3-opt* algorithm considers only the arc exchanges that are feasible. Referring to the figure 2 for each arc (i_1, i_2) that is chosen, the arcs (j_1, j_2) and (k_1, k_2) can be determined by examining a list of the arcs that have i_1 as one end node and the list of arcs that have i_2 as one end node. For each node attached to i_1 and for each node attached to i_2, one of the two nodes in each of the arcs (j_1, j_2) and (k_1, k_2) can be identified. If the necessary pair of arcs exist, then a feasible exchange is possible. If this exchange represents a saving, the exchange is made and the process is repeated. When no more feasible improvements can be identified, the algorithm terminates.

The *3-opt* heuristic algorithm implement in this study uses existing arcs and shortest path whenever the arc to be exchanged does not exist in the original graph. It also utilizes large neighbourhood structure as described in section 3.1. The following is the modified *3-opt* algorithm with the shortest path option, using existing arcs.

The 3-opt algorithm using Existing path, shortest path and enlarged neighbourhood

Use 2-opt improve
Tour = *f(T)*; ImproveA = 0;ImproveB=0
Repeat
If there exists two unvisited nodes **then**
Mark them as visited
Exchange arcs
Else
Find a shortest path to the next unvisited pair of nodes
For k=1 to n **do**
Pick two unvisited nodes j_k, j_{k+1}
For m=3 to n-3 **do**
If ExistArc $((i_m, i_{m+1})$ and (j_m, j_{m+1}) and $(k_m, k_{m+1}))$ then
$delweight = d(i_m, i_{m+1}) + d(j_m, j_{m+1}) + d(k_m, k_{m+1})$
ImproveA=Delweight-$\{d(i_m, j_m) + d(j_{m+1}, k_{m+1}) + d(i_{m+1}, k_m)\}$
Exchange arcs
Else
Find a shortest path to the next unvisited pair of nodes
ImproveS=Delweight $-\{P(i_m, j_{m+1}) + P(i_{m+1}, k_m) + P(j_m, k_{m+1})\}$
EndElse
EndIf
If ImproveS < ImproveA **then**
$f(T') = f(T) -$ ImproveA
Else $f(T') = f(T) -$ ImproveS
Endfor{m}
Endfor{k}
Until (ImproveA=0)and (ImproveS=0) and all nodes Are visited
$Tour = f(T')$

3.3 Iterated algorithm and other approaches

A promising but relatively unexplored idea is to restart near a local optimum, rather than from an initial solution generated by the tour construction heuristics. Under this approach, the next starting solution is obtained from the current local optimum (where the current local optimum is usually either the best local optimum found thus far, or the most recently generated local optimum) by applying a pre-specified move to it. We refer to such a move a perturbation, and to the approach as iterated local search. In this way, not all good characteristics from previously found solutions are lost. Recent research such as Johnson [12], Johnson and McGeoch [13], Martin, Otto and Felten([14],[15]), Martin and Otto

[16] shows that iterated local search can be extremely competitive.

We also considered two other perturbation strategies for the iterated local search for the Sparse TSP. The first approach we considered was to change the distance between cities for a few random cities. This was done to the original problem. Then using the local optimal solution we had, we tried to continue improving the solution.

We implemented an approach of adding a few missing arcs whose arc length were the shortest distance between the nodes. In this approach we were making the graph denser. This approach is based on the work by Codenotti et al ([17], [18]). The improvement in terms of tour quality was reasonable. Making the graph denser produced better results. In order to achieve this, parameter tuning was given a lot of attention. Some of the parameters we used here include the number of arcs whose distances were to be changed, by how much, and how many missing arcs should be added.

We have observed that usually a decrease in the objective function value is considerable in the first steps of the heuristic and then tails off. In particular, it takes a final complete round through all allowed moves to verify that no further improving move is possible. Therefore, in our implementation we stopped the heuristics early after observing a very slow decrease over some period. And we noted that not too much quality is lost.

3.4 Speed-up

Given the proven usefulness of the improvement heuristic algorithms and the many problems to which they have been applied, it is only natural to seek to improve their computational performance while maintaining the quality of the final solution. We aim to reduce the amount of computations done, without letting the solution quality deteriorate too much.

In dealing, with problems similar to large TSP, where a really efficient algorithm for the best solution is unavailable, it is in general time consuming, if not entirely hopeless, to work on refinement techniques to obtain the best solution. Rather, the approach should be to develop a technique by which good locally optimal solutions can be obtained very quickly and with reasonable probability that, among the locally optimal solutions, we may indeed find the best which is near optimal. Sometimes this best solution can be optimal. To get this type of speed-up we used the first improvement.

A straightforward implementation of the k-opt heuristic algorithm, for example, is hopelessly slow. It takes $\frac{n}{k}$ tests to check whether a tour can be improved by *k*-exchange. Even for *k=3*, the heuristic runs almost forever on medium size instances of a few thousand nodes. To make this approach practical, a number of modifications limiting the exchanges considered are necessary. They are based on insights about the probability of success of certain exchanges, or on knowledge about special structures. Well designed fast data structures play an important role as discussed in Fredman et al [19]. The issue of speeding up TSP heuristics is treated in depth in Johnson [12], Bentley [8], and Reinelt [7]. Adopting these techniques and applying them on the Sparse TSP instances of up to a five thousand cities were solved using reasonable computing effort. Hence, most savings in terms of computational cost come from using and designing appropriate data structures particularly those designed to take advantage of the special structures of the applications, employing programming tricks, problem reduction techniques and concentrating search effort on nodes that tend to yield good *2-opt* or *3-opt* swaps.

In this paper the issue of identifying non-optimal arcs has been studied and implemented. Most of the results used in identifying non-optimal arcs are adopted from the work by Jonker and Volgenant [20]. The results are encouraging. We managed to reduce the size of the problems and to restrict the search space.

Lastly, in implementations of our algorithms we limited in advance the number of possible exchange steps to be considered. This leads to deterministic running times and appropriately led to good solutions because the most substantial improvements are usually found early.

4 Computational Results and discussions

Our computational results are obtained for the data from the road network in rural Ireland. We consider problem instances of size between 10 and 4923 cities. For each instance, we ran each improvement heuristic with different starting solutions fifteen times and then took the average. We have also done

testing using data from TSPLIB which have been used in a lot of studies in the literature. Due to space constraints we only report on some of the results, detailed results are found in Mtenzi [21].

Computational results for random starting solutions demonstrate that *2-opt* exchanges are not powerful enough. Results provided in figure 1 show that 2-opt improved a random initial tour on average only by 9%. In his study on large Geometric TSP, Reinelt [7] concludes that random starts are not advisable for the *2-opt* heuristic and that restricted search to speed-up the computations is necessary. Because of the smaller neighbourhood of *2-opt*, it is recommended to use starting tours which are reasonable.

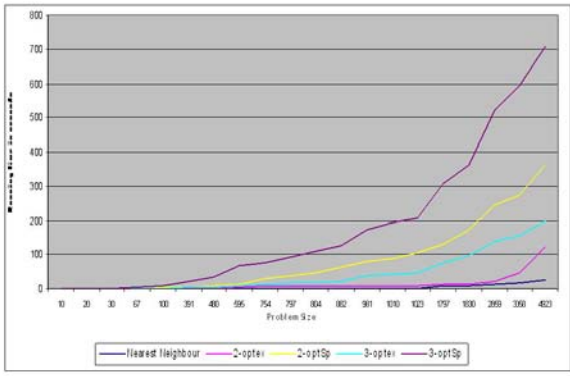

Fig. 1:Solution time for tour improvement heuristics
For problems taken from road network in rural Ireland

The performance of the *2-opt* algorithm varies as a function of the starting tour to which it is applied. Bentley [8] point out that the *2-opt* heuristic algorithm is very sensitive to starting tours. In our computational study in this paper we used the Nearest Neighbour, Farthest Insertion, and Random Insertion tour construction heuristics to provide starting tours for the Sparse TSP. We show that the Farthest Insertion and *2-opt* combination gives the best results.

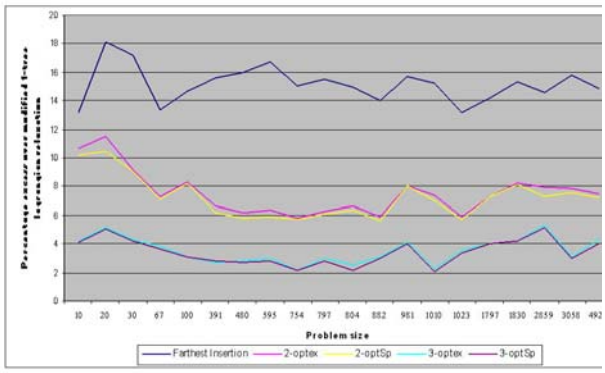

Fig. 2 Solution quality using the Farthest Insertion Heuristic as a starting solution

Having performed a *2-opt* exchange, the direction has to be updated for one of the two segments of the tour. We managed to save some CPU time by maintaining the direction of the longer paths and only reversing the shorter path. This was accomplished by using an additional array giving the rank of the nodes in the current tour. An arbitrary node receives rank 1, its successor gets rank 2 etc. Having initialized these ranks we could determine in constant time which of the two parts is shorter, with the ranks updated only for the nodes in the shorter part.

For problems taken from the TSPLIB, each tour improvement heuristic was run four times on an instance using different starting tours. In this case there was no need to use any of the modified heuristic algorithms designed in this paper because they are/were complete graphs, where all arcs exist. Also due to space constraint we do not include these results, they can be found in Mtenzi [21].

From the results given in figure 2 it can be said that the quality of the final solution that *2-opt* generates strongly depends on the starting solution. This is because *2-opt* is a weaker neighbourhood. On the other hand *3-opt* algorithms are not dependent on the starting solution and produce nearly the same solution quality regardless of the starting solution. This is true for both problem instances we use in this study.

Computational results after including perturbation in our algorithms shows that iterated local search is competitive only if the perturbation is sufficiently large to move a solution outside the neighbourhood of the local optimum. If it is not, then the effect of

the perturbation might be reversed in a single or small number of iterations, and the perturbation would literally lead nowhere. On the other hand we noted that, perturbation should not be too large, or else the good characteristics of the previous local optimum may be lost, and the procedure is then effectively multi-start rather than iterated local search.

Heuristics designed in this study may work with other types of data such as random generated problems. However, considerable amount of redesign may be necessary in some cases, especially in the pre-processing algorithms of the data.

Starting from completely random tours we get a wide sample of all local optima. In problems where we used relatively weaker neighbourhoods, it was crucial to use 'good' starting solutions. Results from our testing uphold this conclusion.

5 Summary and Conclusions

Computational experience of our speed-up techniques discussed in this paper suggests that our improvement heuristics are running significantly fast. However, we have noted deterioration in terms of the tour quality. Nevertheless, using these approaches it was possible to run a large number of experiments within a given time, hence increasing our probability of getting a global optimum tour or better near-optimal tour.

This study has a strong experimental flavour, for the simple reason that, theorems seem to be incapable of adequately characterizing the algorithms. This is so as theoretical results of the Standard TSP are derived with the constraint that the triangle inequality be obeyed. Algorithms designed and implemented in this study are for the sparse graph problem instances in which in some cases the triangle inequality may not necessarily be obeyed. This means that there is a room for more research to be done in the area of theoretical aspects of the Sparse TSP. Moreover, it might be interesting to study the mathematical properties and geometrical characteristics of local optimal produced by our algorithms.

The study of the Sparse TSP is still not finished. Important questions got to be pursued include exploitation of parallel architectures and classification of problem instances to be able to choose automatically the most appropriate algorithm in certain situations, and the most appropriate combination of algorithms. The ultimate goal, of course, is the solution of the Sparse TSP to optimality. But, when claiming to be able to solve practical problems we have to be able to adapt to limits imposed in practice.

Computational experiences obtain from this study highlight the need for using neighbourhoods which are easy to search. At the same time a neighbourhood must be large enough to ensure that the quality of the local optimal obtained is good enough. For large problems the running time of improvement heuristics may, however, be excessive unless sophisticated reduction and data handling techniques are employed. And more important is that the Sparse TSP methods seem to benefit from implementing more than a single neighbourhood.

References:

[1] Lawler, E.L., Lenstra, J.K., Kan, A.H.G.R., and Shmoys, *The Traveling Salesman Problem*, John Wiley and Sons Ltd, 1985.

[2] Johnson, D.S. and McGeoch, L.A., The Traveling Salesman Problem: *A case study, in Local search in Combinatorial Optimization*, Aarts, E. and Lenstra, J.K., Editors., John Wiley and Sons Ltd. pp. 215--310, 1995.

[3] Croes, G.A., A method for solving traveling salesman problems. *Operations Research*, 6: pp. 791--812, 1958.

[4] Lin, S., Computer solutions of the traveling salesman problem. The Bell system technical journal, 1965. 44: p. 2245--2269.

[5] Lin, S. and Kernighan, B.W., An effective heuristic algorithm for the traveling salesman problem. *Operations Research*, 21: pp. 498--516, 1973.

[6] Stewart Jr, W.R., Accelerated branch exchange heuristics for symmetric travelling salesman problems. *Networks*, 17: pp. 423--437, 1987.

[7] Reinelt, G., Fast Heuristics for Large Geometric Traveling Salesman Problems. *ORSA Journal on Computing*, 4(2): pp. 206--217, 1992.

[8] Bentley, J.J., Fast Algorithms for Geometric Traveling Salesman Problems, *ORSA Journal on Computing*, 4(4): pp. 387--411, 1993.

[9] Rego, C., Relaxed tours and path ejections for the traveling salesman problem, *European Journal of operational research*, 2: pp. 522--538, 1998.

[10] Gutin, G., Exponential neighbourhood local search for the traveling salesman Problem.

Brunel University of West London, Department of Mathematics and Statistics, 1996.

[11]Reinelt, G., The traveling salesman: Computational solutions for TSP applications.: Springer Verlag, 1994.

[12]Johnson, D.S. Local optimization and the traveling salesman problem. in ICALP '90. Springer-Verlag, 1990.

[13]Johnson, D.S. and McGeoch, L.A., The Traveling Salesman Problem: A Case Study in Local Optimization. 1995.

[14]Martin, O., Otto, S.W., and Felten, E.W., Large-step Markov chains for the Traveling Salesman Problem. *Complex Systems*, 5: pp. 299--326, 1991.

[15]Martin, O., Otto, S.W., and Felten, E.W., Large-step Markov chains for the TSP incorporating local search heuristics, Operations Research Letters, 11: pp. 219--224, 1992.

[16]Martin, O. and Otto, S.W., Combining Simulated Annealing with local search heuristics, *Annals of Operations Research*, 63: pp. 57--75, 1996.

[17]Codenotti, B., Manzini, G., Margara, L., and Resta, G. Global Strategies for Augmenting the Efficiency of TSP heuristics. 1993.

[18]Codenotti, B., Manzini, G., Margara, L., and Resta, G., Perturbation: An efficient Technique for the solution of very large instances of the Euclidean TSP. Informs Journal on Computing, 8(2): pp. 125--133, 1996.

[19]Fredman, M.L., Johnson, D.S., McGeoch, L.A., and Osteimer, G.O., Data Structures for Travelling Salesmen, *Journal of Algorithms*, 18: pp. 432--475, 1995.

[20]Jonker, R. and Volgenant, T., Nonoptimal edges for the symmetric travelling salesman problem. *Operations Research*, 32(4): p. 837--846, 1984.

[21]Mtenzi, F.J. and O'Connor, D.R., Heuristic algorithms for the Sparse TSP, Business Research Programme, Graduate School of Business, University College Dublin, 1998.