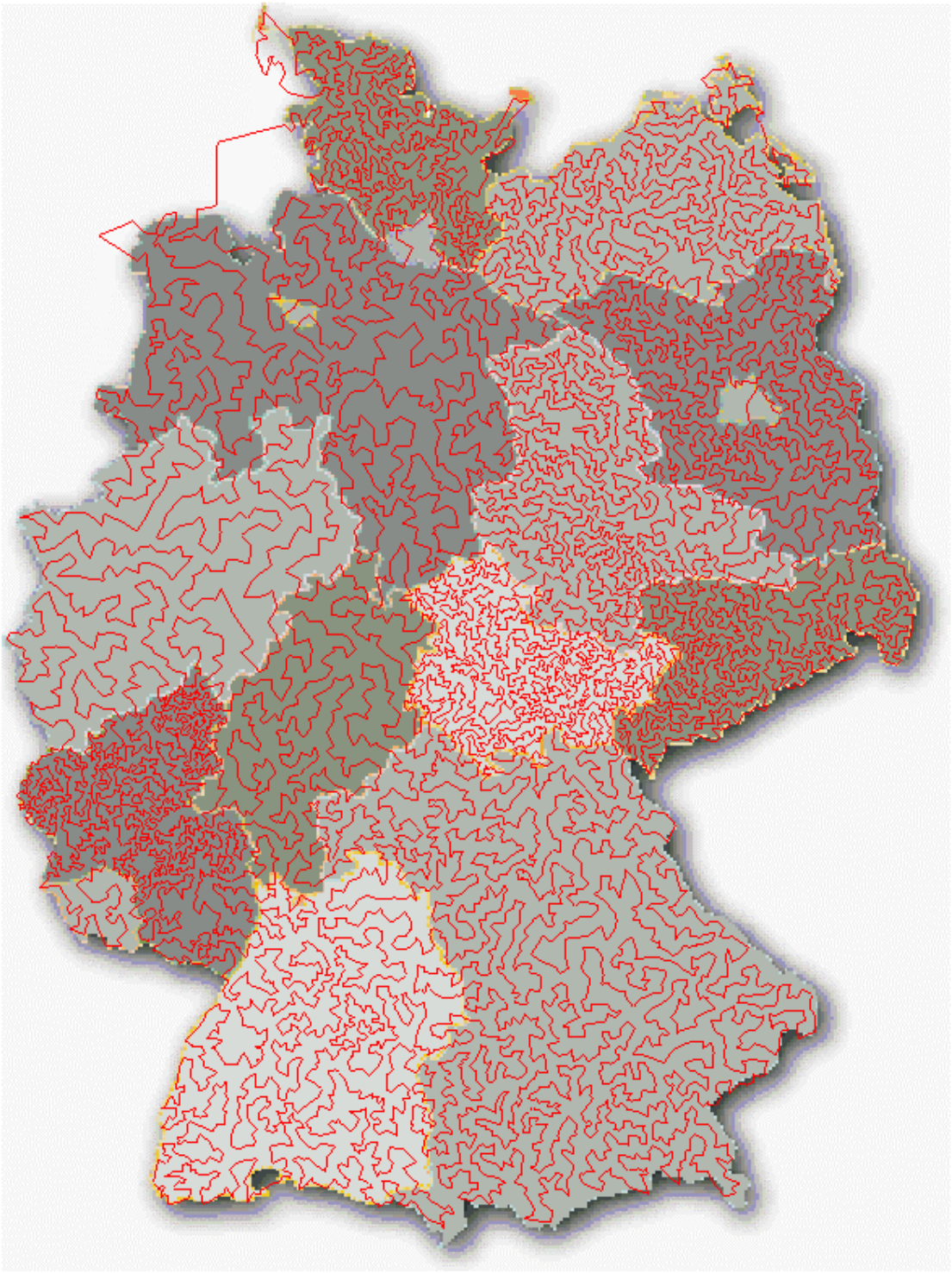
A Multilevel Scheme for the Travelling Salesman Problem

Øystein M. Hjertenes
University of Bergen 2002



Preface:

I would like to thank my tutors Fredrik Manne and Noureddine Bhoumala for support and help. I would also like to thank my girlfriend Line Carlsen for her support and patience, and Per Otto Hjertenes for his assistance in proofreading my texts.

This text is about the Travelling Salesman Problem. An observant reader would already now have discovered that I am using travelling instead of traveling. This spelling difference comes from the subtle differences between U.K English and U.S English. If you are interested in more facts about the name travelling and traveling confusion I would like to refer to the following web site [29], or to [30] where it is also mentioned. The picture on the front page is by Applegate, Bixby, Chvátal, and Cook , and is collected from [31]. The picture shows the optimal tour of the tsp instance d15112 from TSPLIB [25], depicting the shortest tour to travel, if you want to visit all German cities.

Ø. Hjertenes Bergen 2002.

Contents:

Chapter 0 The Thesis

A Multilevel Scheme for the Travelling Salesman Problem

The Travelling Salesman Problem (TSP) is one of the most famous combinatorial problems of all time. A salesman visits *N* cities with given positions and returns finally to his city of origin. Each city is to be visited only once, and the problem is to find the shortest possible route. This problem belongs to a class known as NP-complete problems. In this thesis, we aim at introducing a Multilevel scheme for finding good solutions for large graphs. The Multilevel Scheme works as follows: A sequence of smaller graphs is obtained from the original graphs using a coarsening procedure. Thereafter, an approximate solution to the problem is computed on the coarse graph. This solution is then projected back towards the original graph, by periodically proceeding with an improvement phase using a combinatorial optimisation technique such as the Simulated Annealing method. The objective of the project is to study the impact of different coarsening schemes on the quality of the final solution, and develop a structure for the Multilevel scheme for TSP.

Solving the Thesis.

It is necessary to demonstrate that the Multilevel Scheme (MLS) can be adapted to a TSP environment. Then it is possible to study the impact of different coarsening schemes on the solution quality. To solve these problems, we plan to study the field of approximation heuristics and algorithms for TSP. We will look for schemes and structures in that theory which can benefit us in the creation of the MLS. A clear picture of current use of TSP-solvers also needs to be developed. We will also study techniques for testing the quality of the heuristics we develop.
From this research we plan to develop and test different Multilevel Schemes.
To study and compare the different schemes we propose, we plan to implement them. The solutions the implemented MLS produces will be used to compare them and to locate a best possible combination.

In this paper an introduction to the Travelling Salesman Problem will be present first. Then we are going to present an overview of the methods for solving TSP.
When enough material has been presented to give the reader an understanding of the methods for solving the TSP. Is it possible for us to give a detailed presentation of the Multilevel Scheme, and our approach to it.

The last part of the paper is dedicated to testing the different adaptations and discussing the results found.

Chapter 1 The Travelling Salesman Problem

The object of this chapter is to give an overview of the Travelling Salesman Problem and why it is so interesting to work with. The history of the problem will be presented, and why it is so important to work with solvers for the Travelling Salesman problem will be discussed.

The Travelling Salesman Problem (TSP) is one of the most studied optimisation problem today. TSP is still a case for intensive studies in many fields; one of the reasons for this is that TSP is a NP-complete problem [2]. In The field of algorithms there are many classifications of problems, two of the most important ones are the classes of P and NP. If a problem is in P, it is reckoned that the problem is possible to solve on regular computers within reasonable time. If it is classified as NP as a rule of tomb it is not possible to do so.

One important field of theoretical studies is if a problem classified to be in NP can be solved in such a way, that the two classes P and NP is equal to each other. In NP there exists a set of problems which is call NP-Complete. These problems have the property that if one of those problems can be classified to be in P, all other problems in NP are also in P. Since the TSP is NP-Complete, it is therefore interesting in both the field of theoretical studies of the problem P=NP VS P≠NP, and in the field of approximation heuristics.

To get an understanding of what the TSP is all about it is necessary to define and work with a clear the mathematical representation of the problem.

Definition 1.1: **The Travelling Salesman Problem:**

The general problem can be stated as the following:
If we are given a set of cities $\{c_1, c_2, \ldots c_N\}$ *and for each pair* $\{c_i, c_j\}$ *of distinct cities the distance between them is* $d(c_i, c_j)$*. If a line is drawn trough all the cities visiting each city only once, and end up where it started. The goal is the to find an ordering* π *of the cities that minimizes the total length of that line. This creates a Hamiltonian cycle in the graph of cities.*

Stated more formal:

Using the same notation, the mathematical expression is to minimize:

$$\sum_{i=1}^{N-1} d(c_{\pi(i)}, c_{\pi(i+1)}) + d(c_{\pi(N)}, c_{\pi(1)}) \quad [1]$$

This sum is called the tour length. For the general problem the object is to find one of the paths in the graph that has the shortest length.

This gives us the possibility to formulate the formal language for TSP, which is:

TSP = {(G,c,k) : G=(V,E) is a complete graph,
c is a function from VxV → **Z,**
k ∈ **Z,** and
G has a Travelling Salesman tour with cost at most k}. [2]

This problem is NP-complete [2].
If the P≠NP holds, we need a different problems solving approach to solve problems that are classified to be in NP, then the more direct methods that can be used for solving problems classified to be in P. In *Chapter 2.1* this problem is presented in more detail, and the implications are discussed.

1.1 History of the Travelling Salesman Problem

An Irish mathematician sir W. R. Hamilton and the English mathematician T. P. Kirkman already treated mathematical problems related to the TSP in the 1800's. This work resulted in for example the famous game Icosian by Hamilton. Icosian is the problem of finding a Hamiltonian cycle along the edges of a dodecahedron. I.e. a path such that every vertex is visited a single time, no edge is visited twice, and the ending point is the same as the starting point [3], a game which clearly is not far away from the TSP formulation.

K. Menger [4] seems to be the first Mathematician to write about the general TSP. He showed that the Nearest Neighbour solver was not able to produce the optimal solution of the problem, so he was also one of the first researchers to discuss the hardness of TSP. The first mathematical results of TSP were published in 1940 when TSP solvers first were used as a tool for real life problems. The task was to solve the school buss routing of West Virginia, by Flood [4]. Flood was also involved in the development of other practical uses of TSP solvers in for instance machine scheduling.

Because of factors as algorithmic improvement and hardware improvements over the years, the largest know map with a known optimal solution has grown a lot since the 1950's when the problem was first solved on computers.

Table 1.1 Presented here is a list of historical benchmarks from [5]:

Year	Research Team	Size of Instance
1954	G. Dantzig, R. Fulkerson, and S. Johnson	49 cities
1971	M. Held and R.M. Karp	64 cities
1975	P.M. Camerini, L. Fratta, and F. Maffioli	100 cities
1977	M. Grötschel	120 cities
1980	H. Crowder and M.W. Padberg	318 cities
1987	M. Padberg and G. Rinaldi	532 cities
1987	M. Grötschel and O. Holland	666 cities
1987	M. Padberg and G. Rinaldi	2,392 cities
1994	D. Applegate, R. Bixby, V. Chvátal, and W. Cook	7,397 cities
1998	D. Applegate, R. Bixby, V. Chvátal, and W. Cook	13,509 cities
2001	D. Applegate, R. Bixby, V. Chvátal, and W. Cook	15,112 cities

As one can see the possibilities given us by increased computer power, and the development of new solving techniques, has lead an increase in the size of the problem solved optimality over the past years.

The problems listed in the above table are all in the form of euclidian TSP.
This is a special case of the TSP but is still NP-complete see *Chapter 2.1.*

1.2 The Travelling Salesman Problem Today

Today, because of the ever-expanding computer capabilities in speed and memory, problems with less than 100 cities can be solved to optimality within reasonable time. These problems can of course not be solved with just brute force enumeration and distance checking. For the 16-city Travelling Salesman Problem, the problem of Homer's Ulysses attempting to visit the cities described in *The Odyssey* exactly once, there are 653,837,184,000 distinct routes! [6]. Enumerating all such roundtrips to find the shortest one, took 92 hours on a powerful workstation. By applying test criterions on the fitness of solutions, most routes do not have to be checked, and provide us with a tool to solve Travelling Salesman Problems directly. But unless P=NP proves to hold, a question which by now looks more likely to be P≠NP. No polynomial time algorithm is going to be discovered for the TSP. Therefore it is in the field of approximation algorithms and

heuristics there will be improvements in the future. Today several teams have done approximations on problems with up to a million cities and getting reasonable results [7]. The goals for future research on the TSP must be to optimise existing schemes, and exploring the more general algorithms and heuristics, to bring fort success as in the Travelling Salesman Problem for other fields of optimisation.

1.3 Different Types of Travelling Salesman Problems.

In the field of research on the TSP, there are three main versions of TSP studied. The type of problem depends on how the input function is given: Euclidian symmetric or asymmetric, or random distance matrixes. The term euclidian comes from the representation of each city and the distance of the edges between them. Instead of representing the graph with weighted edges the problem is given as points in the 2-D plane (This can be generalized to problems in K-D for K>2). In order to calculate the distance between points, the euclidian distance formula for points in the plane is used. A euclidian problem is symmetric because the distance satisfies $d(c_i, c_j) == d(c_j, c_i)$. If a problem is asymmetric this equality does not hold. Asymmetric problems studied today are usually a euclidian problem, but with an edge weight in addition on the direction of the travel. Problem-sets where the euclidian distance formula does not hold are called random distance matrix problems. The problem type used by most researchers interested in approximation is the symmetric euclidian TSP. There are many reasons for this; one of the main reasons is that distance can be easily checked. Because it is bi-directional it also saves the programmers a lot of work treating with special cases. This allows for more focus on the development of general heuristics for solving the problem.

1.4 The Reasons for Studying the TSP

Most importantly, the TSP often comes up as a sub-problem in more complex combinatorial problems. It is also very easy to explain and state, but it is intriguingly hard. This has resulted in that the TSP has become one of the main tools in the study of the “P=NP?” problem. Since TSP is so hard and has a lot of real life applications, the field of approximation algorithms and heuristics has bloomed with solvers for the TSP problem. The field has had an enormous expansion since the Nearest Neighbour heuristic where studied by K. Menger[4]. As the field is so large, experience can be drawn from a lot of sources. These sources are one of the important reasons why the TSP is now one of the main test cases for general optimisation techniques.

1.5 Real Life and the TSP

The most fascinating part of the TSP problem is that it often arises in real life problems. This is one of the main reasons for the huge interest in the TSP, and hence the field of real life applications for the TSP deserves a closer look.

The TSP naturally arises as a sub problem in many transportation and logistics applications. For example the problem of arranging school bus routes to pick up the children in a school district. This bus application is of important historical significance to the TSP, since it provided motivation for Merrill Flood, one of the pioneers of TSP research in the 1940s[4]. A second TSP application from the 1940s involved the transportation of farming equipment from one location to another to test soil, leading to mathematical studies in Bengal by P. C. Mahalanobis and in Iowa by R. J. Jessen[4]. More recent applications involve the scheduling of service calls at cable firms, the delivery of meals to homebound persons, the scheduling of stacker cranes in warehouses, the routing of trucks for parcel post pickup, and a host of others.

Increasing the efficiency of processes is of great concern for many industries. It is in this field many commercial applications for a TSP solver can be found. In industry there are a lot of processes involving the time-consuming task of moving items and steering of machinery, which is where a good TSP solver can be used to increase the efficiency.

A Case study: **Circuit board construction and board cutting:**

This industrial process involves a computer-operated machine moving over the 2-D plane adding or cutting parts to or from a flat board. In this process the points where a machine shall operate on the board is plotted as positions in a computer. The computer then has the responsibility of steering the head over the board to the coordinates, performing a set of tasks then returning and a new board is inserted in to the machine. This process is often repeated a large number of times. Each board often has more than 100 operations done to it in different places. A typical example is the process of punching holes in a board for component placement. The route the head of the machine travels over the board can then be visualized as a euclidian Travelling Salesman Problem by setting the coordinates on the board as cities and the TSP tour as the route for the machine to move over the board. A cut in operating time of only 5% is a huge savings cut for a factory, and therefore a TSP solver used to calculate the paths for this problem is interesting. A real life example where this is put in use is at the Metelco circuit board manufacturer where the machine drills 500 holes in each board. Before they used a TSP solver this process took one hour. By using the TSP algorithm this number increased to 4800 holes per hour, and the overall factory output increased by 10% [8]. For a factory this is a substantial increase in production.

Many other fields, both in industrial and within science have found TSP like problems in their field of work. Here are some examples:

Genome sequencing

Researchers at the National Institute of Health in USA have used a TSP solver to construct radiation hybrid maps as part of their ongoing work in genome sequencing. The TSP provides a way to integrate local maps into a single radiation hybrid map for a genome; the cities are the local maps and the cost of travel is a measure of the likelihood that one local map immediately follows another [10].

Starlight Interferometer Program

A team of engineers at Hernandez Engineering in Houston and at Brigham Young University have experimented on using Lin-Kernighan to optimise the sequence of celestial objects to be imaged in the proposed NASA *Starlight* space interferometer program. The goal of the study is to minimize the use of fuel in targeting and imaging manoeuvres for the pair of satellites involved in the mission (the cities in the TSP are the celestial objects to be imaged, and the cost of travel is the amount of fuel needed to reposition the two satellites from one image to the next) [11].

Power Cables

TSP has been used to locate cable placement to deliver power to electronic devices associated with fibre optic connections to homes [12].

DNA Universal Strings

A group at AT&T used a TSP solver to compute DNA sequences in a genetic engineering research project. In the application, a collection of DNA strings, each of length K, were embedded in one universal string (that is, each of the target strings is contained as a sub-string in the universal string), with the goal of minimizing the length of the universal string. The cities of the TSP are the target strings, and the cost of travel is K minus the maximum overlap of the corresponding strings [13].

Concluding this chapter we have seen that the Travelling Salesman Problem has been subject for intensive research over the past decades. There has been a brief presentation of the importance of finding solvers for the TSP. Since the TSP and other problems in NP cannot be solved easily, in the next chapter there will be discussed how to handle a problem that cannot be solved within reasonable time.

Chapter 2 Working With and Testing Solvers for Problems in NP

The object of this chapter is to discuss the implication a classification as a problem in NP has for solving and testing that problem.

Starting by presenting the complexity classes of P and NP, and the importance of NP-Complete problems. Then it is shown where the Travelling Salesman Problem fits in to the picture and presented methods for solving problems in NP. In the last section there will be a discussion on testing the quality of solvers for the Travelling Salesman Problem.

2.1 The Travelling Salesman Problem - a Challenge to Work With.

The Travelling Salesman Problem is as previously stated NP-Complete.
To clarify this a few definitions needs to be presented.

Definition 2.1.1: **P**

P is the complexity class of problems that can be solved in polynomial time on a single tape deterministic Turing machine [9], [24].
If a problem is in P, as a rule of thumb it can be solved within reasonable time on computers existing today. Typical problems of this class are for example sorting problems.

Definition 2.1.2: **NP**

NP is the complexity class of decision problems, for which answers can be checked by an algorithm whit a running time polynomial in the size of the input. Note that this doesn't require or imply that an answer can be found quickly, only that any claimed solution could be verified (or refused) quickly. NP is the class of problems that a non-deterministic Turing machine accepts in polynomial time [9], [24].
If a problem is in NP it has an exponential running time on an ordinary computer and is therefore in most cases not solvable within reasonable time.

Definition 2.1.3: **NP-Complete:**

This is the complexity class of decision problems for which answers can be checked for correctness, given a certificate by an algorithm whose running time is polynomial in the size of the input (that is, it is NP) and that no other NP problem is more than a polynomial factor harder [9], [24].
Informally, a problem is NP-complete if answers can be verified quickly, and if there exist a polynomial time algorithm to solve a NP-Complete problem.
All other problems in NP can be solved in polynomial time. If a problem is NP-complete, as for NP it cannot as a rule of tomb be solved within reasonable time on computers existing today.

The consequences

This implies that if a problem is NP-Complete there is no polynomial time algorithm for it unless P=NP. But this has not been proved yet, and it looks like that probably P ≠NP. If it was possible to find a polynomial time algorithm for an NP-Complete problem, then could by *definition 2.1.3* all NP-Complete problems solved in polynomial time and P would equal NP. But since that question is not proven yet, it is not possible to say there will be such a discovery or there will not be such a discovery.
However it looks like those classes is not equal, and this is where research in this field stand at the moment [24]. Therefore it is not likely to exist a polynomial time algorithm for NP-Complete problems. That means if TSP is NP-Complete there is no known polynomial time algorithm for it, in fact it is O(N!) where *N* is the number of cities.

Proving the NP-Completeness of the Travelling Salesman Problem

The proof of the NP-Completeness of the general Travelling Salesman Problem stated in *definition 1.1* is a classic in the textbooks for the field of algorithmic complexity [2]. This paper is mainly dealing with the euclidian version of the TSP explained in *Chapter 1.3* It is therefore of importance that the euclidian TSP is also NP-Complete.
C. H. Papadimitriou [15] presented the proof for the NP-Completeness of the euclidian TSP problem. Therefore is the current problem NP-Complete, and the following can be assumed:

Solving TSP accurate is not feasible because of the exponential factor O(N!), where *N* is the number of cities. The running times just for solving small instances is enormous, a 30-city tour would be more than $2.65*10^{32}$ operations.

2.2 Working with Problems that cannot be Solved Within Reasonable Time.

Since the task of solving the TSP accurately is not feasible, to get a solution for a TSP problem you could either focus on only small instances, or look for an approximate solution within polynomial time. If you choose to focus only on small instances, you will loose the possibility to solve many interesting problems. One of the reasons for the interest in the TSP is that it often is a part of another problem that can be solved by using a TSP solver. Solving small problems of this type is not often enough since large instances of the TSP problem is related to many industrial and scientific modelling tasks. Another method for solving problems in NP is to find an approximation for the solution.

Approximation algorithms and heuristics

An approximation algorithm finds a near optimal solution to a given optimisation problem, where the solution is never worse than a proven ration.
This ratio is a function of the closeness to the solution [24].

There are two types of approximation algorithms: Maximization and Minimization problems. The goal for an approximation algorithm for the TSP problem is to find the shortest possible route in the graph. Hence TSP is an example of a Minimization algorithm. The quality of an approximation algorithm can be proven, so by using an approximation algorithm it is possible to guarantee a bound on the solution for such problems.

When a method cannot be proven to return a result within a specific ratio it is called a heuristic. The fact that heuristics does not have a worst-case boundary, does not mean it is weaker then the approximation algorithms [1].

An approximation heuristics is a method for problem solving that finds a near optimal solution to a given optimisation problem, where there are no bound on the quality of the solution produced.

For the TSP there are two common types of approximation heuristics: growing and refining.

Growing methods works by constructing a tour from an unordered collection of cities so that the length of the tour is minimized. An example of such a method is a tour construction heuristic.

A refining method takes a pre-made tour and refines its quality. The process of creating tours is easy. For example for an euclidian instance it can be done in O(N) time by picking a start vertex then adding each vertex to the tour in a random order ensuring that no vertex is added more than once. The refinement steps then reorder the tour in such a way that the length of the tour is minimized. An example of such a method is local search heuristics.

The method for comparing the quality of an approximation algorithm is to give a proof for it is worst-case performance. For heuristics that possibility does not exists and you must therefore look at other methods for comparing the effect of the different heuristics. The usual method is to compare how close the solution produced by the heuristic on average, is to the actual solution. This can be a problem as for most cases, the optimal solution is not known.

2.3 Held-Karp and Methods for Testing the Quality of a Solver

The Held-Karp Lower Bound

When evaluating the performance of a TSP heuristics, we are often not allowed the luxury of comparing the results with the precise optimal tour length. Since for large instances, typically the optimal tour length is not known. As a consequence, when studying large instances it has become the practice to compare the heuristics results to something which is possible to compute: The lower bound on the optimal tour length the so called Held-Karp lower bound [14]. The Held Karp bound is defined as the solution to the standard linear programming relaxation of the TSP [14]. For instances of moderate size it can be computed exactly using linear programming. For larger instances there is a problem because the number of constraints in the linear program is exponential in *N* where *N* is the size of the input. Instead a more practical approach is used to solve a sequence of restricted linear programs, each involving only a subset of the constraints, and to use a separation subroutine to identify violated constraints that need to be included in the next linear program. Using the simplex program developed by Applegate, Bixby, Chv'atal, and Cook exact values for the bound have been computed in this way for instances as large as 33,810 cities [14.] For larger instances, you have to settle for an approximation to the Held-Karp bound computed by an iterative te chnique proposed in the original Held-Karp papers [1].
More important is that the Held-Karp bound appears to provide a consistently good approximation to the optimal tour length. From a worst case point of view, the Held-Karp bound can never be smaller than (2/3)OPT(Tour), assuming the triangle inequality [14]. In practice, it is typically far better than this.

Standard Test Instances

When experimental results are discussed in this paper, the problems are symmetric euclidian. Many of the applications of the symmetric TSP are of this sort, and most recent published studies have concentrated on them [1], using two main sources of such instances.

The first source consists simply of randomly generated instances, where the cities have their locations chosen uniformly in the unit square, with distances computed under the Euclidean metric. The second source is a database of instances called TSPLIB collected by Gerhard Reinelt [25] and is available via anonymous ftp from softlib.rice.edu. The problem set can also be downloaded from a range of different web sites.
TSPLIB contains instances with as many as 85,900 cities. Included in the collection are many instances from printed circuit board and VLSI applications as from *the real life example in Chapter 1.5*. The collection also includes geographical instances based on real cities, for example the figure on the front-page of this paper depicting the optimal tour of a graph with 15112 cities in Germany. For this package the Held Karp-lower bound is estimated for all instances and for most of the instances the optimal tour length is also known.
The question then remains if this library of test instances is so diversified that it can be used as a reliable source for testing TSP-heuristics. D.Johnson [1] shows that results where random instances are used as method for comparison are consistent with results where the TSPLIB package is used. Therefore it is no need to use randomly generated sources for testing TSP-heuristics.

Many papers covering geometric instances of the above two types, do not use the Held-Karp bound as their standard for comparison. Researchers dealing with TSPLIB instances, often restrict attention to those for which optimal solutions are known, comparing their results to the optimal tour lengths. Fortunately, the exact Held-Karp bounds are known for all of these instances, so it is easy to translate from one sort of comparison to the other. The results from studies where Random Euclidean instances where they compare their results only to the estimates of expected optimal tour length are more difficult to deal with. D.Johnson [1] shows that many claims of closeness to optimality are too optimistic by 5% or more, and one have to reinterpret such claims for random generated instances.
Thus the TSPLIB is the best test instances for testing the quality of the TSP heuristics.

Chapter 3 Algorithms and Heuristics Used for Solving Travelling Salesman

The object of this chapter is to present an overview of different strategies used to create solvers for the TSP. The object is also to present elements used later in the text for constructing the Multilevel Scheme, and to show how those elements relate to other developed strategies.

TSP Solvers

The work in the field of approximation heuristics for TSP has given us a large set of tools to find possible solutions for the TSP. In this chapter a few of the most used solvers is presented. There are four different classes of solvers used today: tour construction methods, local search methods, backtracking-jump methods, and specials. For each category will we present some of the popular methods used. The heuristics and algorithms are presented as sketches with details where it is needed.

3.1 Tour Construction Heuristics

A tour construction heuristic is a method that from an unordered collection of cities constructs a valid TSP tour.

A tour construction heuristic is usually used to create input for local search heuristics. Most of them can in fact also be used as a fast way to create a reasonable tour. Nearest Neighbour, one of the oldest mentioned was used as early as 1956 [16] for constructing tours for local search heuristics. It is a very easy method to implement and create tour quality on random euclidian sets with reasonable quality even for larger graph sizes [1]. There are many tour construction heuristics. Four interesting ones are presented here: Nearest Neighbour, Greedy, Clarke Wright and Christofides.

3.1.1 Nearest Neighbour

This method is a natural strategy for the TSP, because it mimics the way the travelling salesman selects a travel route.
It selects a starting point and then always selects the nearest city to be added to the tour, it then “walks” to that city and repeats by choosing a new non-selected city, until all cities is in the tour. To complete the tour, an edge is added between the last selected city and the starting city. A general version of this heuristic has running time of $\Theta(N^2)$ [1].

3.1.2 Greedy

This heuristic works by growing a Hamiltonian cycle in the graph.
The Hamiltonian cycle is grown, by first picking the shortest edge, and then the shortest edge available is added to the tour until all cities are included in the tour. An edge is available if adding the edge does not create a loop or a vertex with a degree higher then two, and it is not already a part of the tour. This process is repeated until all cities are part of the cycle and each node in the cycle has a degree of two.
Greedy has a running time of $\Theta(N^2 \log N)$ a bit worse than Nearest Neighbour.[1]

3.1.3 Clarke Wright and Christofides

The two most specialized tour construction heuristics mentioned in this paper are Clarke-Wright (CW) and Christofides. CW is another greedy method that works by choosing a city and then connecting all cites to this city with two edges as if it was a hub. It constructs the tour by removing edges to the hub if it is easier to travel directly from one city to another then trough the hub. This is done until a legal tour is created; the running time for CW is the same as for Greedy [1].

Christofides is the current champion of the tour construction heuristics but it has a far worse running time then the other tour construction heuristics and is not useful on large instances [1]. It works by first constructing a minimum spanning tree *T* for the set of cities, and then a minimum length matching *M* is done on the vertexes with odd degree in *T*. Combining *M* with *T* gives us a connected graph where every vertex has an even degree, this graph now holds an Euler tour [1] i.e. a cycle that passes through each edge exactly once. By first identifying the Euler tour, the TSP tour is then created by traversing the Euler tour.

3.2 Local Search Heuristics and Algorithms

A local search heuristic uses a strategy where solution is taken as input, and then the heuristic samples the solution-space by making simple tour modifications. If the new modified tours are an improvement, it continues to work on it. If it is not an improvement, the modified tour is discarded and the heuristic tries a new modification of the current best tour.

Local search heuristics are specified in terms of a class of operations (exchanges or moves) that can be used to convert a tour into another.

Given a tour, the heuristic then repeatedly performs operations from the given class, as long as each operation reduces the length of the current tour, this process is repeated until a tour is reached for which no operation yields an improvement (a local optimal tour).
It is also an important fact that no local search heuristic can guarantee to make an improvement on the input tour [1].

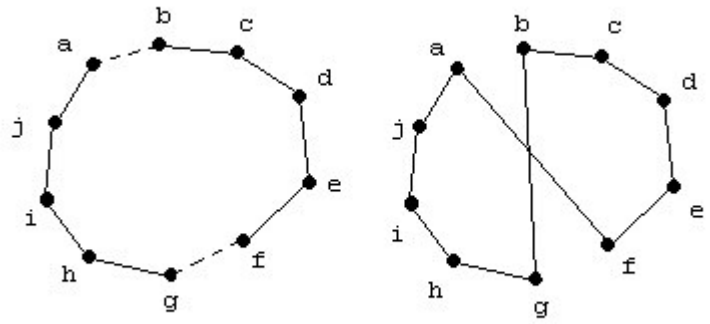
Local search heuristics for the TSP were one of the first methods developed to produce a reasonable tour quality. It includes many famous methods as for example the 2-Opt.

3.2.1 2-Opt

2-Opt is one of the first local search algorithms for TSP. Results regarding 2-Opt was published as early as 1956 [16], where it was used together with the Nearest Neighbour tour construction heuristic.

2-Opt is a simple local search algorithm that works on the basis of doing small changes on the tour and then checking if the solution quality improves.
The change part for 2-Opt is to delete two edges from the tour, creating two tour segments then reconnecting them in a new way so that it forms a correct tour.

Figure 2.2.1 shows a 2-Opt move:

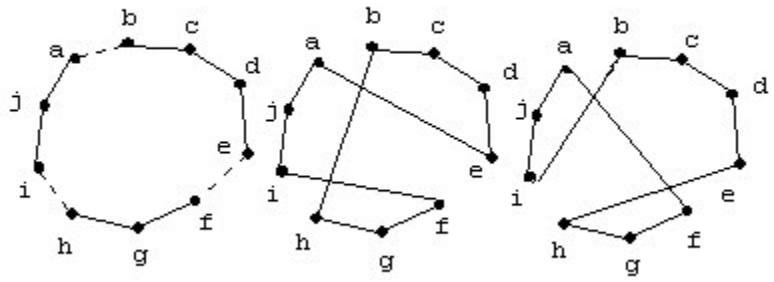


The 2-Opt is run until a stopping criterion is found. The stopping criterion is a test of the possibility to find a new improvement.

3.2.2 3-Opt

3-Opt is an algorithm not unlike 2-Opt, in 3-Opt instead of creating two tour-segments three tour segments are created by removing three edges form the tour. This allows for a new element to be added to the method, it is now possible to reconnect the tour segments in different ways. This gives 3-Opt the possibility to locate the best possible way to reconnect the tour. This checking is the reason for why 3-Opt is a bit slower then 2-Opt, but instead creates tours with higher quality than 2-Opt [1].

Figure 3.2.2 shows possibilities for reconnecting after a 3-Opt split.



Except for the difference in the splitting, the rest of 3-Opt works as the 2-Opt method. These two methods are in fact part of a family of methods called K-Opt.

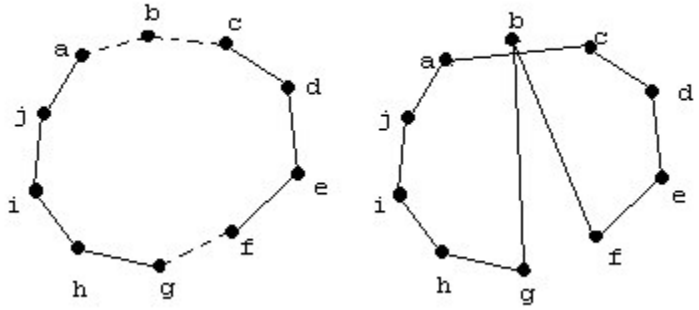
3.2.4 K-Opt variants

In the K-Opt family there are method that lies between 2 and 3-Opt and also methods with K>3. Examples of both classes will be presented later in the text.

Between 2- and 3-Opt

The two methods usually found form this class in the literature are 2.5-Opt and Or-Opt. 2.5-Opt created by Bentley [1] works by moving a city instead of deleting edges. It picks a city *A* from the tour. The method then splits the tour in to two tour segments as in 2-Opt, where two of the end points are the cities in the position before and after *A* and a split in another place of the tour, then it reconnects the tour in the best possible way as for 3-Opt

Figure 3.2.3 shows a possibility for reconnecting after a 2.5-Opt split.



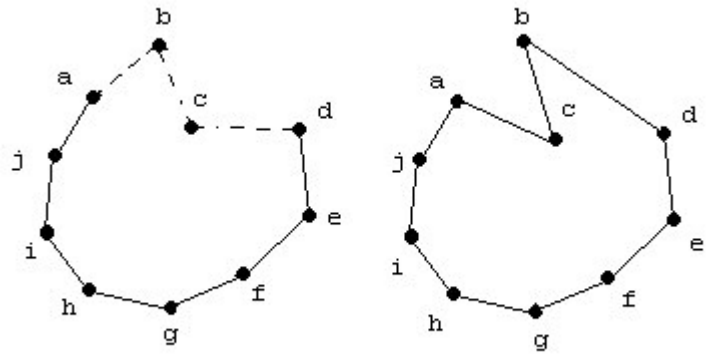
Another method in this class is Or-Opt created by Or[1].
It works by removing a segment with three or fewer cities and then reconnecting the tour as for 2-Opt and adding the line segment between two random neighbour cities.

3.2.5 K-Opt for K>3.

2.5-Opt and Or-Opt are improvements of the 2-Opt method. Some fixed K-Opt, for K>3 has been proposed as improvements of 3-Opt but Lin [1] showed that there was nearly no improvement to be found from these methods.
If a method uses K>3 today, it is used a solver in a variable Opt method.

There are also some methods that lie lower than 2-Opt in the K-Opt hierarchy. These methods are all restrictions on the 2-Opt method. An example of one these methods is the Exchange 2-Opt. In Exchange 2-Opt only neighbour cities are exchanged by moving a city in front of it is neighbour in the tour. This restriction reduces the neighbourhood size to N where N is the number of cites from N*(N-2)/2 which is the neighbourhood size for regular 2-Opt [1]. In practice this restriction seems to be to hard and causes the heuristic to create weaker results then 2-Opt [1].

Figure 3.2.4 shows a possibility for reconnecting after an Exchange-Opt split.



There are also more recently developed local search heuristics. For example Dynasearch by Potts and van de Velde[1] and GENI/GENIUS of Gendrau, Hertz an Laporte [1].

Dynasearch works by doing a set of Opt moves combining them into a single overall move. This effect removes some redundancy from the general Opt strategy.

GENI is a hybrid tour construction and local search strategy. It starts with a tour of 3 cities, and then it proceeds with adding new cities to the tour by doing a 3-or 4-opt split. It then adds the new city to the endpoint of one of the line segments completing the move. Requiring that the edges added is within a neighbour list will help restricting the range of possible insertions, and cut down time for finding local minima.

GENIUS is a local search method that works at the same principles as GENI. It searches the tour for likely candidates to be removed from the tour, removes them and inserts them again using the same method as GENI.

3.3 Backtracking-Jump

Motivated by the observation that not all locally optimal solutions needs to be good solutions, pure local search may not be the answer to the search for the best solvers. It might instead be desirable to modify a pure local search heuristics by providing some mechanism that helps us escape local optima and continue the search further.
A method to do this is for example to give the heuristics the possibility to do reruns on a random generated tour with a local search heuristic storing the best results. This method does not utilize the fact that local optimums is usually clustered together [1], so instead of starting completely anew, a mechanism for climbing out of a local optima looking for a neighbouring one is needed. This idea led to the development of a group of methods, which utilizes the idea in different ways.
The most important tool that heuristic development has given us is a possibility to allow us to make moves that do not improve the tour but may do so in the future.
The improvement in these heuristics is that they allow us to do a series of moves and test if they lead us to an improvement. If the current search does not lead anywhere it is then possible to step back and start anew. The most studied method of this class is Lin-Kernighan (LK)[1]. Tabu Search was one of the first algorithms utilizing this idea [1]. The third presented in this chapter is Simulated Annealing (S.A) a very general optimisation strategy used in many fields of optimisation [1].

3.3.1 Tabu Search:

Tabu search (TS), LK and S.A are motivated by the observation that not all locally optimal solutions are necessary good solutions.

That observation started a process where a range of new methods was created; the first of those was Tabu Search. In general Tabu Search restarts the search near the local optima instead of restarting on a complete newly generated tour.
When TS finds a local optimum, it tries to step out of it by evaluating a series of moves, which picks the best move, but not necessary a move that leads to a better solution.
To stop TS from finding the last optimal solution it keeps a list (a taboo list) over the moves that would lead back to an old local optimum. When no new moves are possible it stops.

3.3.2 Lin-Kernighan:

The Lin-Kernighan (LK) algorithm is generally considered to be one of the most effective methods for generating optimal or near-optimal solutions for the TSP. However, the design and implementation of LK is not simple. There are many designs and implementation decisions to be made, and most decisions have great influence on the performance. The creation of the LK was inspired by the observation that a static *K* in the K-Opt method is not necessary the best solution. Designers wanted to use a different K-Opt in different stages in the execution of the heuristic. In practice it has been shown that it is difficult to know what *K* to use to achieve the best compromise between running time and quality of the solution [1]. Lin and Kernighan removed this drawback by introducing a powerful variable-Opt algorithm. The algorithm changes the value of *K* during its execution [1].

Before each iteration, the algorithm decides what values of *K* should be used. It calculates the value of *K* by examining for ascending values from l, whether an interchange of *K* links may result in a shorter tour. Given that the exchange of *r* links is being considered, a series of tests is performed to determine whether *r+1* link exchanges should be considered. This continues until some stopping conditions are satisfied. At each step the algorithm considers a growing set of potential exchanges (starting with $r = 2$). These exchanges are chosen in such a way that a feasible tour may be formed at any stage of the process. If the exploration succeeds in finding a new shorter tour, then the actual tour is replaced by the new tour.

3.3.3 Simulated Annealing:

Simulated Annealing is a Monte Carlo approach for minimizing functions with a large number of variables. The term Simulated Annealing comes from a roughly analogous chemical process of heating and then slowly cooling a substance to obtain a crystalline structure. In simulation, where the cost function is minimized, this corresponds to when the substance have crystallized. The Simulated Annealing process lowers the temperature in stages until the system “freezes” and no further changes occur. At each temperature the simulation must proceed long enough for the system to reach a steady state. This process is known as thermalization. The sequence of temperature decrements, and the number of iterations applied to thermalize the system at each temperature comprise an annealing schedule [23]. To use Simulated Annealing, the system is first initialised with a particular configuration. A new configuration is constructed by displacing the system in to a new state. If the energy of this new state is lower than that of the previous one, the change is accepted unconditionally and the system is updated. If the energy is greater, the new configuration has a probabilistic chance to be accepted. This is the Metropolis step, the

fundamental procedure of Simulated Annealing. The Metropolis step allows the system to find areas with lower energy states, and yet still jump out of local minima due to the probabilistic acceptance of some upward moves.

Simulated Annealing as a method for solving optimisation problems was first proposed by a team of computer scientist from IBM in 1983 [17]. They proposed to use it as a computing intensive algorithm for finding solutions to arbitrary optimisation problems.

Now it has been used successfully as a solver for a wide range of fields.

As a TSP solver it has been use successfully to solve problems to near optimum. Simulated Annealing as a TSP solver is presented in *Chapter 5.*
Johnson and McGeoch in [1] shows us that Simulated Annealing in his implementation is not as good as LK but has a good average and a lot of room for interesting adaptations. They also state that the Simulated Annealing structure in use today is not necessary an optimal one, and there can be made interesting adaptations to it. It mentions temperature and solution neighbourhood as interesting fields for experimenting. Especially low temperature starts where Simulated Annealing is given a high quality starting solution.

3.4 Other Methods

TSP research has also givens us some more exotic solvers. The two methods worth mentioning are Genetic heuristics and iterated local search.
Genetic heuristics as an approach to optimisation can be tracked back to the 1970's. The heuristic applies the theory of the biological process of evolution to a problem. The heuristic has two basic steps. First, create a group of many random tours ordered in a population. These tours are stored as a sequence of numbers. Second, pick two of the better (shorter) tours as "parents" in the population and combine them, using crossover to create two new solution children in the hope that they create an even better solution. The crossover step is performed: By first selecting a random point in the parent sequences, and then switching every element in the sequence after that point. Then the good solutions are allowed to reproduce and form new and hopefully better solutions in the population, while the bad solutions are removed until a stopping criterion is reached.
Iterated local search algorithms are a very interesting adaptation of the local search algorithms. It has been for example seen used by the team that develop the Concorde TSP solver [22] in their iterated LK version. The essence of this scheme is: one iteratively builds a sequence of solutions generated by the algorithm, leading to far better solutions than if one were to use repeated random trials of that algorithm. ILS explores the search of local minima for some given algorithm, called Local Search in the following.

ILS achieves this by doing the following: Given the current best solution S^*, it first applies a change or perturbation that leads to an intermediate state S' from the set of possible solutions S. Then local search is applied to S' and it reach a solution $S^{*\prime}$. If $S^{*\prime}$ passes an acceptance test, it becomes the next element of the walk, otherwise one returns to S^*. This ILS procedure should lead to good sampling as long as the perturbations are

neither too small nor too large. If they are too small, one will often fall back to *S** and few new solutions will be explored. If on the contrary the perturbations are too large, *S'* will be random, and the result is a random restart type heuristic.

Because TSP is one of the most interesting testing grounds for optimisation heuristics and algorithms, there are many solvers for it. Comparing all of them is enough material for a life time study. But it should also be mentioned that a lot of work has been done in the field of parallel programming on some of the above-mentioned methods. But describing these are outside the scope of this text.

Chapter 4 The Multilevel Scheme:

In this chapter the generic Multilevel Scheme is presented, together with a study on how to adapt it to the Travelling Salesman Problem. The object of this chapter is to give the reader a good understanding of the Multilevel Scheme. The parts that need to be developed in the Multilevel Scheme for a given problem are discussed. The last par of the chapter will present how to develop the elements in a MLS for the TSP.

4.1 The Generic Multilevel Scheme

In this paper, another way to look at heuristics is presented: The Multilevel Scheme. The Multilevel Scheme is not a new idea. It has been applied with success on other problems for example graph partitioning [18], and graph drawing [19].
The idea behind the Multilevel Schemes is to coarsen a problem in such a way that the original problem is easier to solve. The coarsened solution is then in a series of steps refined until the original problem size is reached. The effect of each refinement step is propagated to the refinement in the next level. This produces in theory a high quality solution after the top step refinement.

The Benefits produced by the MLS structure for Optimisation Problems.

In optimisation problems are often solved, by using refinement on a constructed solution. But to construct a good solution from the problem to use in refinement can be as hard as the problem it self. A low quality input solution to a refinement method produces inn fact also poorer quality after the refinement [1]. The size of the problem is also important; it is usually much easier to find a good solution on a small problem then on a large problem.

The Multilevel Scheme utilizes this property by refining on small problems, producing a good solution for the small problem. Then it expands the high quality solution in to a new problem where the quality found in the smaller problem size is also propagated up in to the larger problem. The next refinement step can therefore benefit by getting a better-input solution.

Presenting the Generic Multilevel Scheme

Let us present the generic Multilevel Scheme, and then let us do some analysing on it.

Name:	**The Generic Multilevel Scheme**
Input:	Problem instance P_0
Output:	Solution S_0 on P_0

1 **for** i← 1 to L

1.1 P_i = **coarsen** (P_{i-1})

2 $S_{L.u}$ = **initialize** (P_L)

3 S_L = **refine**($S_{L.u}$)

4 **for** i←L-1 to 0

4.1 $S_{i.u}$ = **extend**(S_{i+1})

4.2 S_i = **refine**($S_{i.u}$)

A problem *P* is in the heuristic coarsened in a predefined number of levels *L*. The number of levels can be calculated in advance so that the solution on the bottom level is not just trivial, or the coarsening can be set to stop before the problem size is trivial. The fully coarsened problem is then initialised to a form a refinement algorithm for the problem can work with. This unrefined solution $S_{L.u}$ is then refined by the refinement algorithm to S_L. The heuristic then step vice extend the problem, and refine it until the full problem size is reached and the last refinement step produces the solution S_0.

The coarsen step:

In the coarsen step the object is in some way, to decrease the size of the problem step vice. But it is important that the problem is not trivialized, and it is still intact so that the refinement on the lowest stages can be propagated up in to the next levels.

The initialise step:

The object of the initialise step is to prepare the coarsened solution in such a way that it can be used in a refinement solver for the problem.

The extend step:

The problem size is in this step increased again. The increasing should be done in such a way that the quality of the refinement in the previous step is propagated up a level.

The refinement step:

The refinement step is a regular refinement algorithm for the problem type.

The Multilevel Scheme (MLS) must also hold some more conditions:

Condition 1: Any solution in the coarsened stage should be a legitimate solution on the problem type. Therefore the heuristic should produce a legitimate solution if it was only initialised and then extended without refinement to full size.

Condition 2: The coarsened problem should reflect the original problem, so that the problem space is not destroyed. Therefore the coarsening has to be done in such a way that we adapt the coarsened problem, to reflect the original problem. By letting the new coarsened problem inherit the hardness of the original problem fulfils this condition.

The generic Multilevel Scheme above tells us nothing of how to construct all the parts needed to create a Multilevel Scheme for a given problem. For each different optimisation problem where the Multilevel scheme is applied, we need to develop methods to tackle each of the above-mentioned parts of the heuristic.

4.2 Adapting the Multilevel Scheme for the Travelling Salesman Problem

To fulfil all the above-mentioned conditions, three important parts and some smaller adaptations needs to be identified and constructed. First a coarsening heuristic has to be constructed; it needs to fulfil the conditions that the input graph is not reduced to triviality. An extension heuristic has to be constructed, so that no improvement is lost from the past refinement steps. For refinement algorithm there are already many available versions mentioned in *Chapter 3*, which can easily be adapted. An initialising method also needs to be constructed and the whole scheme has to fulfil the multilevel conditions.

The Coarsening Step

Our proposed method for coarsening TSP type problems is to decrease the number of cities in the input graph. But for the method to hold condition 1 and 2, this cannot be done just by randomly removing cities. The method for coarsening we choose to use is to collapse edges by creating a cluster of the two cities with the collapsed edge.

But first we need some definitions

A *map* is the input graph to a euclidian Travelling Salesman Problem.
A *city* is a coordinate on map; it has an x coordinate and a y coordinate.
A *cluster* is a collection of two or more cities, where the cluster is represented as a city on the map. The coordinates of the cluster are modified to reflect the collapsed cities within it by calculating an average position. This condition is included to full fill condition two.

The coarsening heuristics for the TSP we have designed has two characteristics.

1. Looking for edges that have a high probability to be in the optimal solution.

2. Removing edges that are easy to fit in to the tour later, and thus creating a coarsened problem with the most difficult edges yet to place.

The reason for including the first characteristic is to locate edges that have a high possibility of being in the optimal solution, and therefore the edges have a large chance for not needing to be changed after the extension of the solution.
The object of this method is to produce a good solution after the extraction and therefore decrease the work needed in the refinement step. These heuristics can be constructed by using the principles of already existing tour construction heuristics because they work by the same principles.

The second characteristic is to collapse edges that are easy to place in each step. Thus the coarsening creates a solution with the most difficult edges to locate when the problem is coarsened fully, and the difficulty to locate and place edges decreases therefore for each extend step, while the size of the problem increases.
Inspiration for such characteristics can be found in methods for excluding possible moves in optimisation heuristics.

The Initialization Step

The initialization of a coarsened problem into a TSP type problem is not difficult for our coarsening scheme. Since the output of the coarsening is a map, it is already a TSP type problem. The only initialization we need to do is to transform the instance of type map into a form our refinement algorithm can work with. This is typical to construct a TSP tour on the map. Such a tour can be produced by just picking a random start cluster then at random pick a new cluster to add to the tour until all clusters are included in the tour. Another method is to use an already existing tour construction heuristic on the clusters. The use of a tour construction heuristic can possibly create a better input for the first refinement step then just a random generated tour.

The Refinement Step

There is no need to create a new refinement algorithm for TSP. There are already many well-tested refinement algorithms described in textbooks [1].
Some of them may need some minor adaptations to work with the MLS.

The Extension Step

The extending of the problem after the refining is an important part of the scheme.
The object is to propagate the quality from the previous solution to the next refinement step. To do this the extending heuristic has to keep the order of the tour more or less intact after the clusters has been split and the tour has been extended.
Therefore the collapsed edges are now reintroduced in the position of the cluster in the tour. The method for extension then works by locating every cluster that shall be opened in the current level of expansion then splitting them by reintroducing the collapsed edge to the tour.

Has this Description of an Adaptation of the TSP Heuristic Fulfilled the MLS Requirements?

This scheme fulfils clearly the requirements for the general MLS.
Then the only thing left is to argue that the two conditions are held.

Condition 1

The point of condition 1 is that if one does not use a refinement method, the MLS should produce a legitimate solution of the problem. The constructed scheme clearly manages to do this. A tour is constructed in the initialising step. The extension heuristic for each new split of a cluster splits the tour in the position of the old cluster. On each end the two new cities from the collapsed edge is added. The tour is then reconnected again by adding the edge between those two cities. This operation clearly does not add any loop or fragments. Therefore the resulting tour is still a valid tour after each splitting. The resulting tour is therefore still valid after it has been fully extended. So it is possible for us to conclude that condition 1 holds.

Condition 2

The point of condition 2 is that the coarsened problem should not lie to far away from the original problem. This condition is held because for each cluster reflects the collapsed edge after merging. The coarsened solution therefore inherits the hardness of the original problem, and holds therefore condition 2.

So this scheme for the MLS holds. The heuristics for each part are presented in detail in *Chapter 5.*

Direct Coarsening versus Recursive Coarsening

There is still an aspect yet to be discussed before presenting the heuristics.
It is the question of Direct versus Recursive coarsening.
The general MLS heuristic presented in the preceding chapter is for Recursive coarsening. This method creates clusters for each level by merging two clusters in to a new cluster. Another method to reduce the problem size is to use Direct coarsening [18]. This scheme has nearly the same strategy as the Recursive coarsening except it differs in the coarsening part. Instead of clearly levelled coarsening, Direct coarsening lets the clusters grow until they reach a predefined size and then evict the cluster from the possible choice for merging. This does not damage any of the above-mentioned restrictions. So Direct coarsening also holds the MLS conditions.
The heuristics for both versions and the more subtle differences are explained in detail in *Chapter 5.*

4.3 The Elements that needs to be Decided and Developed in our Study of the MLS for the TSP:

To ease the creation of methods a general template for coarsening and extension needs to be developed. Then using the general template, it is possible to develop a set of tools to use in both classes. When this work is finished, we need to create the to different MLS: Direct and Recursive.
Now we are ready to develop the different elements that shall be included in the two different MLS. The only previously mentioned coarsening heuristic for TSP was a method described in [19]. So to get inspiration for coarsening heuristics we will study the field of tour construction and local optimisation for ideas that can be turned into good coarsening heuristics.
The last part studies how to integrate the refinement heuristics. Walshaw has used the refinement method Lin-Kernighan in an MLS already [19], by making an interface to CONCORDE [22]. Therefore it would be interesting to experiment if the MLS would work with another approximation strategy for TSP. We chose to use Simulated Annealing, since it is an interesting method, and has as stated before still many possibilities for optimisation [1].
Since S.A has so many variables that need to be optimised, for S.A to sample the solution space correctly. This part will therefore needed considerable work in testing and experimenting. The algorithms and heuristics for each of these parts are presented in *Chapter 5* and *Appendix 2*. Results are presented and discussed in *Chapter 6* and *Chapter 7.*

Chapter 5. The Multilevel Scheme for the Travelling Salesman Problem

In this Chapter we are going to present the heuristics constructed by us, and modified for use in the Multilevel Scheme (MLS) for the TSP. We will start by giving a detailed description of the Direct and Recursive coarsening schemes. For each scheme all necessary methods are presented in detail. Then we are going to present the Simulated Annealing and the adaptations and possible optimisations we plan to use. The object of this Chapter is to present an abstract and detailed view of the methods. The levels of details in the methods are pseudocode. Details that are not important for the methods are not presented. Included is also a worst-case running time analysis for each of the different parts developed. Detailed description for each of the important heuristics and algorithms used in our program can be found in *Appendix 2*.

5.1 Introduction

Before of each MLS is presented, some common details found in every method needs to be defined.

5.1.1 Notation

A MAP is an unordered collection of cities (clusters). A TOUR is an ordered collection of the cities (clusters). Both MAP and TOUR has sizes, the expression SIZE(element) returns the size. A *city* or a *cluster* is a point in the plane with a pair of coordinates and is always denoted with big letters. The coordinates of a city is denoted within brackets behind the city, so if A is a city $A(x)$ and $A(y)$ is its coordinates.
When it is important to display the content of a cluster the notation is a combination of the internal cities. If A is merged with B to a cluster it will be denoted as AB. The cluster coordinate is denoted in the same way as city coordinates.

5.1.2 Merging and Extraction of Clusters

In the scheme constructed by us, graphs are coarsened by collapsing edges. Merging cities in to clusters is done by removing the connected cities from the map, and then adding the new cluster to the map instead. If this scheme shall hold the multilevel condition 2, the coordinate of the cluster needs to be updated to reflect the merged cities within. When two cities or clusters are merged an average coordinate is calculate based on the coordinates of the involved cities or clusters. This new average coordinate is now set as the coordinate for the cluster, which is added to MAP. The two cities or clusters are then removed from MAP.

When the problem is extended after refinement the clusters needs to be split in to the two cities which where merged. This process called extraction is the direct reversal of merging. But to hold the multilevel restriction, extraction must ensure that the cities are removed from the cluster in the correct order. If a city is removed from the cluster too early, all the assumptions done in the coarsening are destroyed. These two processes is from now on called MERGE(ELEMENT, ELEMENT) and EXTRACT(ELEMENT), where ELEMENT is a cluster or a city.

5.2 The Recursive and Direct Multilevel Scheme

The general structure for the MLS was presented in *Chapter 4.* In this Chapter the two heuristics: Recursive and Direct is presented in detail.

5.2.1 The Recursive Multilevel Scheme

Recursive coarsening was the first coarsening scheme described. Recursive coarsening in our scheme creates clusters with an internal structure like a binary tree. This structure makes it easy to keep track of the order of extraction from the clusters.
The extraction of the Recursive scheme is easy since the clusters are arranged as a binary tree. Splitting is done by locating the root, then removing the root node creating two new trees where the roots are added back to MAP. This can be repeated until there are only trees with no leaves left in the tour. This solution tackles the problem with the order of extraction, and clearly holds the conditions mentioned in the description of the extraction.

Name:	RECURSIVE MLS
Input:	MAP, number of coarsening levels.
Output:	Finished TOUR.

1 **for** i←0; i<number of coarsening levels; i++

1.2 Prepare MAP for merging

1.3 **Coarsening_heur** (MAP)

2 Initialize by creating a tour with the fully coarsened MAP.

3 Improve the tour with **Refinement_alg** (TOUR)

4 **for** i←0;i<numlev;i++

4.2 **extract a level**(TOUR)

4.3 Improve the tour with tour **Refinement_alg** (TOUR)

Comments:

Input: *number of coarsening levels.*

This number is determined by calculating how many times the graph needs to be coarsened to obtain the problem size we wish to use.

Line 1.2: prepare MAP for merging

This line includes all the processes involved in transforming the input and the active MAP into a format usable by the coarsening methods. The work involved in this line depends on the implementation of the different coarsening methods.

Line 1.3: coarsening_heur (MAP)

This is the general coarsening heuristic. The running time of this step is dependent on the heuristic that has been used.

Line 2: Create a tour on the bottom level.

Since output from the coarsening is already of the TSP form, the method only needs to initialize the output from the coarsening to an input usable by the refinement method. Therefore the running time should be no more then for one of the tour construction heuristics presented in *Chapter 3.*

Line 3 and 4.3: Improve the tour with tour refinement_alg (TOUR)

In this process a standard tour refinement heuristic is used. The running time of these steps is therefore dependent on the running time of the heuristic picked.

Line 4.2: Extract a level (TOUR)

In this step the extraction principle is used and the running time is therefore dependent on the method used.

Running time analysis:

The running time for the heuristic clearly depends of the coarsening depth and running time of each of the elements. Since it is important for the total heuristic to have a non-exponential running time, it must be ensured that each part is non-exponential in time. Therefore each part created by us must be analyzed, to ensure that no parts run in exponential time.

5.2.2 The Direct Multilevel Scheme

Direct coarsening is the second MLS we are going to study. It does not create the tree structure seen in the Recursive heuristic. Instead it creates clusters with a predefined largest size, in our scheme that size is defined to be on average four. In this scheme the internal cities of clusters points to each other in circles instead of trees. The same coarsening heuristic used for Recursive coarsening can be used for Direct coarsening except for a few differences. When a merging has been done the clusters are added back to the map, instead of removed. When a cluster grows to the predefined size, the cluster is set as non-valid choice for merging. All other aspects are the same as before, the only difference is that the pointers within the clusters are arranged as a list and not as a tree structure.

Name:	DIRECT MLS
Input:	MAP
Output:	Finished TOUR.

1 Prepare MAP for merging

2 **Coarsening_heur**(MAP)

3 Create a tour on the bottom level.

4 Improve the tour with tour **refinement_alg** (TOUR)

5 **for** i←0; i< Size(Cluster); i++

    5.1 Extract a city from each cluster in TOUR

    5.2 Improve the tour with tour

    refinement_alg (TOUR)

Comments:

Line 1: prepare MAP for merging

This line includes all the processes involved in transforming the input and the active MAP into a format usable by the coarsening methods. The work involved in this line depends on the implementation of the different coarsening methods.

Line 2: coarsening_heur (MAP)

This is the general coarsening heuristic. The running time of this step is determined by the heuristic that has been used.

Line 3: Create a tour on the bottom level.

Since output from the coarsening is already of the TSP form, the method only needs to initialize the output from the coarsening to an input usable by the refinement method. The running time should therefore be no more then for one of the tour construction heuristics presented in *Chapter 3.*

Line 4 and 5.3: Improve the tour with tour refinement_alg (TOUR)

In this process a standard tour refinement heuristic is used. The running time of these steps is therefore dependent on the running time of the heuristic picked.

Line 5.1: Extract a city from each cluster in TOUR

In this step the extraction principle is used and the running time is therefore dependent on the method used.

Running time analysis:

The running time for the heuristic clearly depends of the coarsening depth and running time of each of the elements. Since it is important for the total heuristic to have a non-exponential running time it must be ensured that each part is non-exponential in time. Therefore we have to analyses each part created by us, to ensure that no parts run in exponential time.

5.3 Coarsening

In the next chapters we will present the heuristics for each part of the two schemes.
In this chapter will first present the two general coarsening methods. Then we will discuss ideas and methods used in the selection part of the two coarsening heuristics.

5.3.1 Recursive Coarsening:

Recursive coarsening takes as input a MAP and returns a MAP half its size.
The object of the coarsening is to locate edges that fit a predetermined criterion and remove them from the MAP. The coarsening method does this by using a selection method; the method selects the edges and uses the MERGE method to remove them from the MAP. It is important that this is done in non-exponential time so that the overall time of the MLS is still polynomial.

Name:	GENERAL RECURSIVE COARSENING
Input:	MAP
Output:	A coarsened MAP
1. Initialise MAP 2. **While** still able to find merging partners 2.1 Do merge selection of A and B 2.2 **MERGE**(A,B) 2.3 Remove A and B from MAP 2.4 Add AB to MAP 2.5 Set AB as not available for merging 2.6 Update selection info	

Input:

This method gets MAP in a state of coarsening as input.

Line 1: Initialise MAP

Some methods need a special rearrangement of the MAP. For example creating a set of possible edges to choose from. The running time of this method is dependent on the selection type.

*Line 2: **While** still able to find merging partners*

This loop is executed at most N/2 times where N is the number of cities in input

Line 2.1: Do merge selection of A and B

In this step the method chooses the merging partners. The running time is depends on the selection method.

Line 2.2 to 2.5

These steps are merge steps and are constant time operations.

Line 2.6: Update selection info

This is the information required by some of the selection methods. The running time depend on the selection method.

Running time analysis:

The total running time is dependent on the method of selection. No part mentioned here is exponential.

5.3.2. Direct Coarsening:

The Direct scheme is similar to the Recursive scheme in most parts.
The same selection methods for Recursive coarsening can be used for Direct coarsening. But there are three major changes to the method:

1. The coarsening is not run in levels but until no more merging can be done.
2. When a cluster is created it is not evicted from the possible merge partners, it is added back to MAP and set as available.
3. When a cluster has grown in to a predefined size, it has to be set as not available as a choice for merging.

Name:	GENERAL DIRECT COARSENING
Input:	MAP
Output:	A completely coarsened MAP

1.Initilaise MAP

2. **While** still able to find merging partners

  2.1 Do merge selection of A and B

  2.2 **MERGE**(A,B)

2.3 Remove A and B from MAP

2.4 **If** AB< max-cluster-size

2.4.1 Add AB to MAP

2.4.2 Update

2.5 **else**

2.5.1 Add AB to MAP

2.5.2 Set AB as not available for merging

Line 1: Initilaise MAP

Some methods need a special rearrangement of the MAP. For example creating a set of possible edges to choose from. The running time of this method is dependent on the selection type.

Line 2: ***While*** *still able to find merging partners*

This loop is executed less than N times where N is the number of cities merged.
Since for each execution of the loop one city is removed and one cluster is added back.

Line 2.1: Do merge selection of A and B

In this step the method chooses the merging partners. The running time is dependent on the selection method.

Line 2.2 to 2.5:

All these steps treat the merging operation and are constant time operations. The exception is *Line 2.4.2* where the arrangements for the selected step are updated. The running time of that line depends on the selection method.

Running time analysis:

The total running time is dependent on the method of selection. No part mentioned here is exponential.

5.3.3 Selection

The important part of the coarsening methods is the selection of merging partners.
To find good selection methods, we studied previously used ideas in tour construction heuristics and other selection ideas. From this basis a set of selection methods was decided on for us to implement and test. In this chapter a description of each selector is presented together with the idea behind the selector.

Random

This method selects two random partners and merges them. The idea is to create a test method to see if the different selection methods have an effect. If the other selection methods produce a solution with higher quality than Random, then it is the selection method that creates the improvement.

Time analysis:

Recursive:

Random does not need any special ordering of MAP to function.
This gives us a total running time of O(N). The method can be found in *Appendix 2*.

Direct:

There is no difference between the Direct and the Recursive structure for this method.
The update function is not needed and therefore the running time is O(N).

Nearest Neighbour

This heuristic was inspired by the heavy edge merging of N. Bouhmala [18].
The object of heavy edge merging is to find the current best available choice in the problem space. It was also inspired by the Nearest Neighbour tour construction heuristic which is a parallel to the heavy edge method. The selection method picks at random a cluster from MAP. It then locates the nearest available cluster on the map and merges them.

Time analysis:

Recursive:

The selection loop is executed N/2 times. The check for finding the nearest neighbour is executed N-I times for each iteration of N where I increase with 2 for each time the loop is executed. This gives us a total running time of order $O(N^2)$, where N is the number of cities.

Direct:

The selection loop is executed N times. The check for finding the nearest neighbour is executed N-I times for each iteration of N where I increase with 1 for each time the loop is executed. This gives us a total running time of order $O(N^2)$, where N is the number of cities.

Greedy

This selection heuristic is an adaptation of the Greedy tour construction heuristic [1].
The Greedy tour construction heuristic is explained in *Chapter 3.1.2.*
This heuristic merges the two still vacant closest cities on the map.

To be able to find the closest cities fast, the method needs to use a distance matrix. Since the input is of the type symmetric Euclidian instances, the distance matrix is upper triangular and can be generated in $O(N^2)$ time.

When the distance matrix is generated it is possible to create a fast lookup table. The lookup tables stores the closest cluster, for each cluster, together with the distance between the clusters.
Then when two clusters are removed from the MAP, it is easy to update the system, by locating all the clusters with a shortest edge to one of the two clusters removed, and then update those entries.

Time analysis:

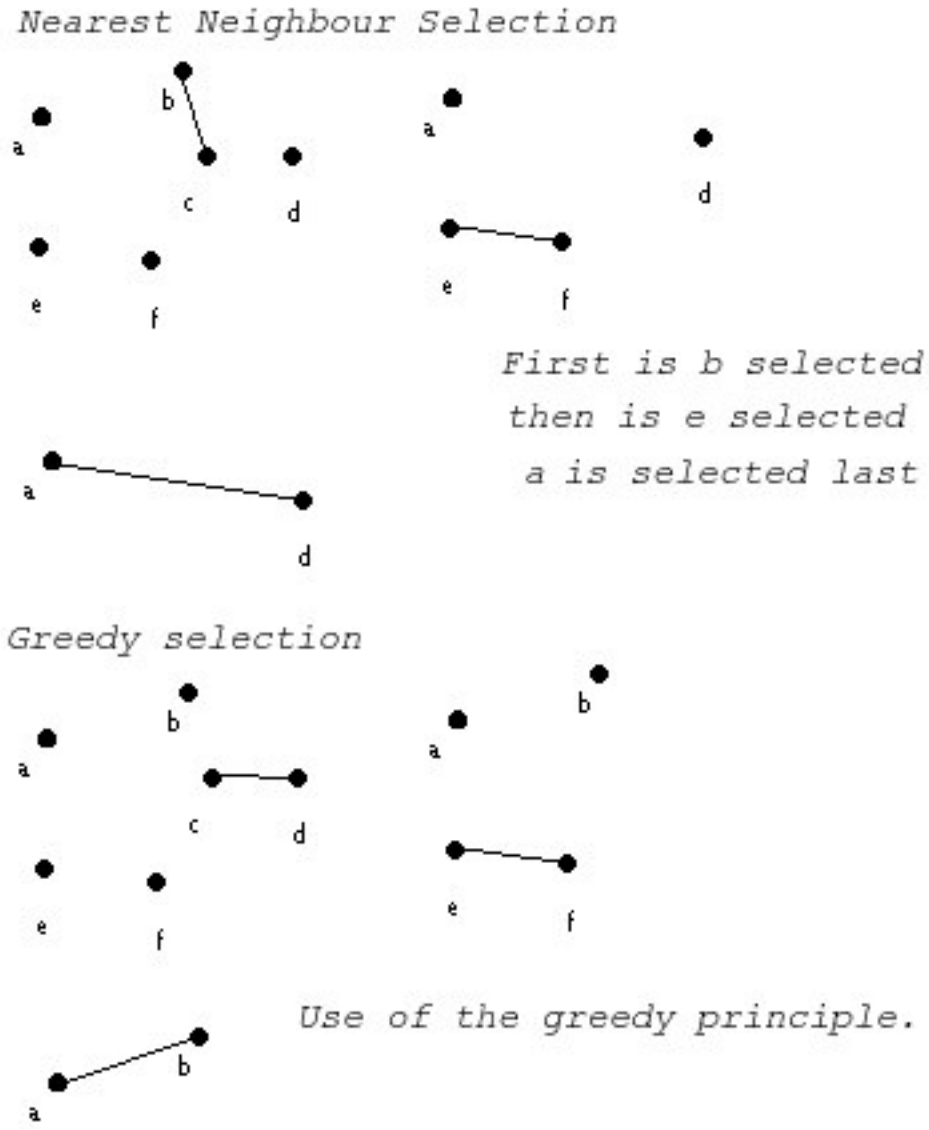
Recursive:

Generating a distance matrix is done in $O(N^2)$ time. Creating the Shortest edge List is done in $O(N^2)$ time. The selection loop is executed N/2 times. The selection is executed for N-I times where I increase with 2 for each execution.
Where N is the number of cities.
Update has a running time in K*N where K is the number of edges to be updated and K is usually much smaller then N, but it is possible to have a value close to N. This gives us a total running time of $O(N^2*K)$. It is also possible by using a heap structure for each node to get a running time of $O(N^2 \log N)$, but it is not necessary worth the extra implementation work, since K is usually much smaller then N.

Direct:

Generating a distance matrix is done in $O(N^2)$ time. Creating the Shortest edge List is done in $O(N^2)$ time. The selection loop is executed less than N times. The selection is executed for N-I times where I increase with 1 for each execution.
Where N is the number of cities.
The rest of the method behaves as the Recursive version and the running time analysis for that part is as for the Recursive version.
This gives us a total running time of $O(N^2*K)$. As for the Recursive version it is possible to use a heap structure for each node to get a running time of $O(N^2 \log N)$. But as for the Recursive version it is not necessary worth the extra implementation work.

Figure 5.3.3.1 Shows how the Nearest Neighbour and the Greedy selection functions.



Max Min

This selection method is not unlike the Greedy selection; this strategy is to minimize the sum of the length of the edges by selecting always the longest “shortest edge” not yet merged.
It uses the same data structure as Greedy. But instead of selecting the shortest “shortest edge” it selects the longest “shortest edge”.

Time analysis:

Recursive:

Generating a distance matrix is done in $O(N^2)$ time. Creating the Shortest edge List is done in $O(N^2)$ time. The selection loop is executed N/2 times. The selection is executed for N-I times where I increase with 2 for each execution.
Where N is the number of cities.
Update has a running time in K*N where K is the number of edges to be updated and K is usually much smaller then N but it is possible to have a value close to N.
This gives us a total running time of $O(N^2*K)$. It is also possible by using a heap structure for each node to get a running time of $O(N^2 \log N)$ but it is not necessary worth the extra implementation work, since K is usually much smaller then N.

Direct:

Generating a distance matrix is done in $O(N^2)$ time. Creating the Shortest edge List is done in $O(N^2)$ time. The selection loop is executed less than N times. The selection is executed for N-I times where I increase with 1 for each execution.
Where N is the number of cities.
The rest of the method works as for the Recursive method.
This gives us a total running time of $O(N^2*K)$. By using a heap structure for each node is for this method also possible to get a running time of $O(N^2 \log N)$.

Square

This is the selection scheme used by C.Walshaw in [19].

Square selection works by ensuring that long edges are not merged in the upper levels of the ML scheme. The idea for this method is that long edges have the highest possibility of not being a part of the optimal tour in a uniformly spaced TSP input. It works by partitioning the map in to squares, where each square contains an average number of cities. When the square is calculated for the heuristic, it is assumed that the cities are uniformly spaced. C. Walshaw proposed to have an average number of the cities equal to ten in each square. From those two notions it is possible to calculate the width and the height of the squares with the square formula.

Definition: **Square Formula**

$h = \sqrt{(An/N)}$.

Where A is the area of the smallest rectangle containing all the cities, n is the average number of cities per square, and N is the number of cities on the map.

As the graph is coarsened, h will increase because there will be fewer cities on the map. Therefore it needs to be recalculated for each level of matching. In the last coarsening there will be only one square.
To position cities within the squares a bucket sort adaptation is useful. Each bucket is a square of MAP. Then it is easy to calculate from the coordinate of each cluster to locate which bucket it goes in to. The selection inside each square is executed by finding the closest city within the square or in one of the adjacent squares.

Time analysis:

Recursive:

The calculation of h is done in constant time. Sorting in to the buckets takes O(N) times. The selection loop is executed N/2 times. The selection is executed for N-I times where I increase with 2 for each execution. This gives us a running time of $O(N^2)$

Direct:

This selection scheme is not usable in Direct coarsening. The size of the squares must be updated as the size decreases. Leading to a lot of work in resorting and updating, together with the problem of when to recalculate the squares. Because of this and since it is not one of our selection methods; choose we not to implement the Square for the Direct scheme.

Radius

This is an adaptation of the Square selection strategy, it is an attempt to create an easy, and on average, fast scheme. It works by using the formula for calculating the length of the edges *h* in Square (see the Square method). This creates a circle around each cluster with diameter of *h*. The heuristic selects a random city or cluster and merges it with the first found cluster or city within this circle. This scheme provides us with a way to ensure that no long edges are merged in the first levels of merging.

Time analysis:

Recursive:

The constant *h* can be calculated in constant time. The selection loop is executed no more then N/2 times. The selection is executed less then N times, where N is the number of cities, but can be as large as N.
This gives us a worst case running time of $O(N^2)$.

Direct:

The constant *h* can be calculated in constant time, and since this operation is constant, it is possible for *h* to be calculated between each selection. The selection loop is executed no more then N times. The selection is executed less then N times, but can be as large as N. This gives us a worst case running time of $O(N^2)$, where N is the number of cities.

5.4 Initialize

When MAP is fully coarsened it must be initialized to a form usable for the refinement method we have chosen. Since it was chosen to use Simulated Annealing (S.A), the coarsened map must be transformed in to a form usable by S.A. S.A takes as input a TSP tour and refines it. Therefore problem the output from the coarsening must be transformed in to this form. Since the product after the coarsening is on the same form as a symmetric euclidian input, using a standard tour construction heuristic to initialize the output can therefore fulfill this criterion. The running time is therefore no more then the running time for the chosen tour construction heuristic. Where N is the number of clusters when MAP is fully coarsened.

5.5 Extension and Extraction Heuristics.

The extraction of a cluster is the process of splitting a cluster in two lesser parts. This creates an edge with two vertexes to be inserted in the tour. To ensure that the quality of the solution from the last refinement step is not damaged in the extension, the new edge must be inserted in the tour position of the split cluster. Since this is now an edge with two vertexes, there are two possible ways to add the edge to the tour, see *figure 5.5.1*. To insert the edge in one of the two different ways shown in *figure 5.5.1*, one can either just use a random method to choose on of the possibilities, or use a local optimisation method to ensure that the quality form the last refinement step is propagated to the next refinement step.

Figure 5.5.1

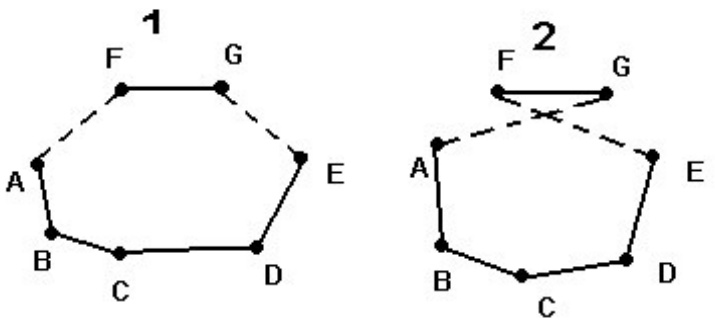


Figure 5.5.1 shows two different possibilities for inserting the edge in TOUR.

Both of the extraction methods are presented here. There are no real difference between Direct and Recursive extension. The only difference is the handling of cluster pointers and will not be commented in this text.

But first let us present the general extension scheme:

Name:	GENERAL EXTENSION
Input:	A refined TOUR
Output:	An extended TOUR
1. **for** all clusters 1.1 **SPLIT**(CLUSTER) 1.2 Insert edge in TOUR 1.3 Remove CLUSTER from MAP 1.4 add the two cities to MAP	

Comments:

Input: *A refined TOUR*

The input to the extension method is the refined TOUR. And it is therefore not necessary to initialise this input.

*Line 2. **for** all clusters*

All current clusters need to be split. The method searches the tour and splits the current clusters. Since all clusters needs to be split, this loop is executed N times where N is the number of clusters in input.

*Line 2.1 **SPLIT**(CLUSTER)*

This is the reversal of the **MERGE** operation and is a constant time method.

Line 2.2 Insert edge in TOUR

Inserts the edge in the TOUR at the position of the cluster. Two methods has been created for this, both have constant time operation, so that step has a constant running time

Line 2.3 to 2.4

Updates MAP and are constant time operations.

Time analysis:

All lines except line 2 are constant time operations which gives us a running time of O(N) where N is the number of clusters at input.

Edge insertion:

Two different edge insertion methods RANDOM and BEST-FIT shall be tested.
The object behind the test of different edge insertion methods is to experiment if it is possible propagate better the quality from the past refinement step.

RANDOM

This method chooses randomly one of the two ways to insert edges, as depicted in *figure 5.5.1*.

BESTFIT

This method minimizes locally the length of the new line segment added to TOUR.
In the figure this method would minimize the length of the path from E-A.
By locating the shortest path of the two different possible paths in *figure 5.5.1*.
It then inserts the edge F-G in the setting that minimizes this line segment.

This calculation is also a constant time operation, but has a constant factor higher then RANDOM.

5.6 Refinement

We now move on to present the refinement method. Since it is important for the comprehension of our experiments to understand the refinement method; Simulated Annealing is present here in detail. Because Simulated Annealing do not have an exponential running time and it is not one of our designs, there will not be presented a running time analysis for it.

5.6.1 Simulated Annealing Generally

The principle behind Simulated Annealing was explained in *Chapter 3.3.3.*
In this chapter the Simulated Annealing (S.A) adapted to TSP is presented.

The principle of the TSP version of S.A is very like the principle of a normal local search strategy. It uses permutations of the input tour to look for improvements.
The permutations used are inn fact adaptations of local search methods.

But S.A is an improvement of the local search strategy used. Since S.A has the ability to do moves that are not an improvement of the tour, but can be so in the future.

S.A starts with an input tour *S.* It then creates a copy of the tour *S** which S.A sets as the best current tour. We now have two tours *S* and *S*.* One to work with *S,* and one where it store the best possible version found *S*.*
S.A then starts to make permutations of the tour and creates a temporary *S'* . When a permutation is made, it compares the resulting length with *S*,* if it is of higher quality then *S** it discards *S** and sets *S'* as *S** and *S.*

If it is not the best current solution, it is not discarded as in a local search method. S.A now checks if it shall continue to work on the weaker solution. Each move has a percent chance to be approved if it is not the best current solution. If it accepts the move, *S* is discarded and *S'* is set as *S,* and S.A continues to work on this new tour. Or else it discards *S'* and makes a new permutation on *S*. The percent chance for acceptance of a bad move is lowered as the solution "cools".
This goes on until the solution has "frozen", when a bad move has a very minute chance to be accepted.

Now let us present Simulated Annealing.

Name:	SIMULATED ANNEALING
Input:	TOUR
Output:	Refined TOUR

1. Generate a starting solution S and then set the initial champion solution S* = S.
2. Determine a starting temperature T.
3. **While** not yet frozen do, the following:

3.1 **While** not yet at equilibrium for this temperature, do the following:

3.1.1 Choose a random neighbour S' of the current solution.

3.1.2 Set Δ = Length(S') -Length(S)

3.1.3 **if** $\Delta \leq 0$

3.1.3.1 Set S = S'

3.1.3.2 **if** Length (S)<Length (S*)

3.1.3.2.1 Set S*=S

3.1.4 **else**

3.1.4.1 choose a random number *r* uniformly from [0,1].

3.1.4.2 **if** $r < e^{-\Delta /T}$

3.1.4.2.1 Set S=S'

3.1.4 End "While not yet equilibrium" loop

3.2 Lower the temperature T.

3.3 End "While not yet frozen" loop.

Comments:

Line 1

In the Multilevel Scheme the starting solution is the previous expanded TOUR.

Line 2

To determine start temperature a formula suggested by N. Bouhmala [21] was used
T_init = -Δlength(*S*)avg /log(*P*),

Where Δlength(*S*)avg is calculated by repeatedly doing iterations on *S* and calculate the average of the changes.
P is the probability for a accepting a bad move (an uphill move),
P values is from 0.999999->0.00000. A *P* value of 0.80 gives an uphill move an 80% chance of being accepted.

Line 3

The loop in this line has a stop condition for when the solution temperature is equivalent to a frozen solution. The temperature for the frozen solution is set as a static value that can be lowered if the solution still has possibilities for improvement.

Line 3.1

The number of executions of this line is defined as the search length. This is the number of changes, which can be performed to a tour before it is rejected.
This number needs to be adjusted. It is also dynamic in the sense that it checks if there has been an improvement. If it has been improved within a factor, it does not lover the temperature, else if no improvement was found the temperature is lowered. This step is the main contributor to work in the S.A, and a large number for search depth and therefore leads to a considerable running time.

Line 3.1.1

This is the process of making a permutation on the TOUR. This can be done in many ways; usually by using a local search strategy.

Line 3.1.3.2

If a new champion solution is found, the champion solution is updated.

Line 3.1.4.2

This line checks if the bad move is accepted and the work TOUR is updated with this bad move.

Line 3.2

The lowering of the temperature is executed by multiplying the current temperature with a value from 0.9 to 0.999. Most authors recommend 0.95.

Line 3.3

In this line the not frozen loop is ended with a dynamic check on if the solution has frozen. The method has a value where it is anticipated that the solution is frozen. When the temperature reaches this value, S.A starts to check if the solution is frozen, by testing if there has been an improvement in the last runs of the search loop. It continues to run while the solution still improves. This value is set in [21] to be 2^{-5}.

5.6.2 S.A Adaptations to the Multilevel Scheme

Since the quality of the input tour before the refinement from Simulated Annealing (S.A) is of high quality, S.A can be started on a much lower chance for accepting an uphill move than a random input. This saves a lot of running time since the temperature does not have to be lowered so often.
But the best start value for the percent chance of accepting an uphill move, can only be found by experimenting with the settings. This is one of the many optimisations S.A needs before it can work properly with our MLS.

Optimisations:

As mentioned before the correct percent chance of accepting uphill moves has to be located. The others are:
Choosing the correct permutation method and optimising the length of the search path.
Testing the effect of the dynamic search depth.
All these optimisations is explored in *Chapter 6.*

5.6.3 S.A Permutations

To find tour improvement, S.A uses permutations. It is a simple scheme. It changes the tour in some random way and compares the new length with the old length and checks if it improves the quality of the tour. Many permutation schemes has been used and proposed, usually it is a K-Opt move. The main difference between them is the number of edges changed. One of our objects in this paper is to test different permutation methods. The reason is that the different permutation methods have been shown to not behave in the same manner [1], and it is important to find one that suits our task.

Therefore it was chosen to test different methods for permutation. Four different permutations have been chosen to use in the experiment, all of them are adaptations of the local refinement methods described in *Chapter 3*: 2-Opt, 3-Opt, 2.5-Opt and Exchange 2-Opt.

5.7 Comparing Our Scheme with Chris Walshaws Scheme.

Since C.Walshaw already has shown that it is possible to create a MLS [19] for the Travelling Salesman Problem, it is interesting to compare our scheme with his to see where we fit into the picture.
Our Recursive scheme is modelled after C.Walshaws scheme. As refinement method C.Walshaw used LK and chained LK instead of Simulated Annealing. He implemented a multilevel interface to the perfected TSP solver Concorde[22]. The Square method described in our text, is modelled after the same coarsening method used by him. The object of C.Walshaws paper was to show that it was possible to create a MLS for TSP. Compared with our goal to test different schemes and different coarsening methods, C.Walshaw also focused on testing the quality produced by ordinary LK and different settings of chained LK with Concorde.

Chapter 6 Experimental Results

In this chapter the results obtained by testing the different Multilevel Schemes will be presented and compared. The object is to check if the different MLS is returning results, and which elements developed for the MLS is best to combine. Included is also the testing of the different adaptations developed for Simulated Annealing. The best settings found is then used to create a reference table, comparable with other results for TSPLIB instances.

6.0 Starting the Experiments

First some initial information concerning the experiments needs to be presented before the results can be studied. This chapter explains the details used in the experiments.

System settings:

All test runs were done on a pc Pentium III with an 800Mhz processor and 128 MB ram, using the Windows ME operating system and the Dev-C++ Compiler/Debugger. The program was written in the programming language C. All test instances used is from

TSPLIB [26], ranging in size from 10^3 to $2*10^4$ cities. All test instances used can be found in *Appendix 3*, where size and optimal length for each instance are given.

The Parameters

As mentioned before, there is a range of different parameters that needs to be optimised in this experiment. It is always best to start working with optimal parameters, but since this is not an option for us; some of the parameters must be predetermined.

Search length

The search length was set to a value of 2N where N is the number of cities in the current step. That value was chosen since we would like to start with a reasonable fast running time.

Refinement

As a refinement method the 2.5-Opt was chosen since we wanted to investigate the performance of this method.

Temperature decrement factor

The Temperature decrement factor is set to 0.99. Since we hope that this value will give us finer resolution, when we test methods that perform nearly identical.

Chance of accepting move at start

Was set to a value recommenced in [21] 30%.

Selection Method

The Nearest Neighbour (N.N) selection method was selected for coarsening. Since the principle behind this selection is well tested in other TSP studies, it should therefore perform well for our use.

Method of extraction

This is the parameter that needs to be tested first, since it can have an important effect on the rest of the scheme. Therefore there is no initial method set here.

6.1 Effect of Different Extraction Methods on the Extension Step

Two different methods for extraction has been developed, see *Chapter 5.5*. Since the extension step is important for the experiment, we have to be sure that we use the best possible method for this step. The two different methods was BESTFIT and RANDOM.

Table 6.1.0 *Experiment settings:*

Parameters	***Value***
Search length	2N
Refinement	2.5-Opt
Temperature decrement factor	0.99
Chance of accepting move at start	30%
Selection Method	Nearest Neighbour
Method of extraction	Varies

We choose to use a set of different test instances for this experiment. The two extraction methods on different instances after each step of extraction are then compared. The object is to see if there is a difference in the quality after each extension step. The table uses normalised values; they are obtained by taking the length produced by RANDOM divided by the length produced by BESTFIT. This produces an easy to understand table for comparison, where 1 is equal. If the value is larger then 1; RANDOM produces a longer tour after extension. If the value is less then 1, RANDOM produces a length shorter then BESTFIT.

Table 6.1.1 *Compares the two different methods for extraction of a set of graphs after each extraction.*

Step:	pr1002	d1291	u2319	rl5915	brd14050	average
6	1.291	1.197	1.147	1.200	1.175	1.202
5	1.188	1.203	1.140	1.183	1.149	1.173
4	1.243	1.154	1.154	1.114	1.170	1.167
3	1.219	1.195	1.162	1.160	1.160	1.179
2	1.182	1.158	1.167	1.165	1.146	1.163
1	1.116	1.144	1.163	1.174	1.181	1.156
0	1.164	1.183	1.169	1.188	1.174	1.175

Conclusion:

From the table it can easily be seen that RANDOM extraction produces results with lower quality than BESTFIT. Since after each step of extraction on all instances the result by using RANDOM is worse the results from BESTFIT.
RANDOM produces inn fact tours that are always in average 1.15 times longer than BESTFIT. Therefore the use of the BESTFIT method propagates better the quality from the past refinement step. The difference is also nearly equal on all steps for each instance.

6.2 Coarsening Size for Recursive Coarsening

Since it is important to test when the coarsening for the Recursive scheme should be halted, we choose to test different coarsening sizes on different tables. The idea is that by reducing the size of the problem to 10% would be sufficient. We compared the results obtained by reducing it first to 20% and then to 50%.

Table 6.2.0 *Experiment settings:*

Parameters	***Value***
Search length	2N
Refinement	2.5-Opt
Temperature decrement factor	0.99
Chance of accepting move at start	30%
Selection Method	Greedy
Method of extraction	BESTFIT

In this normalized table we have used the result from our assumed reduction to 10% by dividing the results from the other settings by the results produced by reducing it to 10%.

Table 6.2.1 *Compares the coarsening size for Recursive coarsening*

Name:	pr1002	rl5915	pla7397
20%	1.005	1.017	1.014
50%	1.015	1.138	1.127

From the table it is obviously that the reduction to 10% produces better results then both of the two other sizes. But the quality increase obtained by reducing the number of cities

on the MAP from 20% to 10% is too small to let us believe it is useful to reduce the instances any more. We can therefore conclude that it is enough to coarsen the instances to a size where only 10% percent of the cities are active.

6.3 Comparing the Different Methods of Selection for Recursive Coarsening

The development and testing of the different methods of coarsening, is one of the main goals of this paper.
The experiment has three parts:
Testing different selection methods for Recursive coarsening
Testing different selection methods for Direct coarsening methods
Comparing Recursive coarsening with Direct coarsening.

Testing different selection methods for Recursive coarsening is best done by comparing the results from runs on the same instance with different selection methods.

A set of test instances where selected, and as for the previous experiments normalized values is used.

Table 6.3.0 *Experiment settings:*

Parameters	***Value***
Search length	2N
Refinement	2.5-Opt
Temperature decrement factor	0.99
Chance of accepting move at start	30%
Selection Method	Varies
Method of extraction	BESTFIT

In these tables the normalized values are calculated by taking the result value from the row and divide it by the value in the column. The tables are therefore best read row by row. As for the last experiment 1 is equal, larger then 1 is longer and less then 1 is shorter.

Table 6.3.1 *Compares the different methods of selection with the Recursive coarsening scheme.*

Name: pr1002	N.N	Greedy	Min Sum	Square	Radius	Random
N.N	1.000	1.002	0.983	0.968	0.833	0.584
Greedy	0.998	1.000	0.981	0.966	0.831	0.583
Min Sum	1.017	1.019	1.000	0.985	0.847	0.594
Square	1.033	1.035	1.015	1.000	0.860	0.603
Radius	1.200	1.203	1.180	1.162	1.000	0.701
Random	1.712	1.715	1.683	1.657	1.426	1.000

Table 6.3.2 *Compares the different methods of selection with the Recursive coarsening scheme.*

Name: rl1323	N.N	Greedy	Min Sum	Square	Radius	Random
N.N	1.000	1.028	0.975	0.760	0.793	0.501
Greedy	0.973	1.000	0.949	0.740	0.772	0.487
Min Sum	1.025	1.054	1.000	0.780	0.813	0.514
Square	1.051	1.080	1.025	1.000	0.834	0.527
Radius	1.261	1.295	1.229	0.958	1.000	0.631
Random	1.996	2.051	1.947	1.518	1.584	1.000

Table 6.3.3 *Compares the different methods of selection with the Recursive coarsening scheme.*

Name: u2319	N.N	Greedy	Min Sum	Square	Radius	Random
N.N	1.000	1.010	1.015	0.956	0.839	0.597
Greedy	0.990	1.000	1.005	0.946	0.830	0.591
Min Sum	0.985	0.995	1.000	0.941	0.826	0.588
Square	1.046	1.057	1.062	1.000	0.877	0.625
Radius	1.193	1.204	1.211	1.140	1.000	0.712
Random	1.674	1.691	1.700	1.600	1.404	1.000

Table 6.3.4 *Compares the different methods of selection with the Recursive coarsening scheme.*

Name: rl5915	N.N	Greedy	Min Sum	Square	Radius	Random
N.N	1.000	1.029	0.949	0.939	0.719	0.361
Greedy	0.972	1.000	0.922	0.913	0.699	0.351
Min Sum	1.054	1.084	1.000	0.990	0.758	0.380
Square	1.065	1.096	1.010	1.000	0.765	0.384
Radius	1.392	1.431	1.320	1.307	1.000	0.502
Random	2.773	2.852	2.631	2.604	1.993	1.000

Table 6.3.5 *Compares the different methods of selection with the Recursive coarsening scheme.*

Name: pla7397	N.N	Greedy	Min Sum	Square	Radius	Random
N.N	1.000	1.032	0.996	0.971	0.785	0.301
Greedy	0.969	1.000	0.965	0.941	0.761	0.292
Min Sum	1.004	1.036	1.000	0.975	0.788	0.302
Square	1.030	1.063	1.026	1.000	0.808	0.310
Radius	1.275	1.315	1.269	1.237	1.000	0.384
Random	3.323	3.427	3.309	3.226	2.607	1.000

Table 6.3.6 *Compares the different methods of selection with the Recursive coarsening scheme.*

Average:	N.N	Greedy	Min Sum	Square	Radius	Random
N.N	1.000	1.023	0.976	0.910	0.783	0.437
Greedy	0.978	1.000	0.954	0.890	0.766	0.428
Min Sum	1.025	1.048	1.000	0.933	0.802	0.448
Square	1.045	1.069	1.019	1.000	0.817	0.456
Radius	1.282	1.311	1.250	1.166	1.000	0.555
Random	2.451	2.511	2.393	2.251	1.903	1.000

Conclusion

The test experiments from the selection methods for Recursive coarsening show us that the result of each of the methods had better quality than the Random selection method. This clearly indicates that the different selection methods have an effect. Greedy produces clearly the best results since on nearly all instances it produces better results than the other methods. Nearest Neighbour is good second with MinSum and Square producing nearly equal results. Radius clearly produces a worse tour than the other methods but it still works better then the Random selection.

6.4 Comparing the Difference Between the Methods of Direct Coarsening.

The goal of this experiment is the same as for the Recursive coarsening.
We wish to find the best overall selection method to be used with our MLS

Table 6.4.0 *Experiment settings:*

Parameters	***Value***
Search length	2N
Refinement	2.5-Opt
Temperature decrement factor	0.99
Chance of accepting move at start	30%
Selection Method	Varies
Method of extraction	BESTFIT

A set of test instances where selected, and as for the previous experiments normalized values is used.

In these tables the normalized values are calculated by taking the result value from the row and divide it by the value in the column. The tables are therefore best read row by row. As for the last experiment 1 is equal, larger then 1 is longer and less then 1 is shorter.

Table 6.4.1 *Compares the different methods of selection with the Direct coarsening scheme.*

pr1002	N.N	Greedy	Min Sum	Radius	Random
N.N	1.000	1.026	1.006	0.900	0.607
Greedy	0.975	1.000	0.980	0.878	0.592
Min Sum	0.994	1.020	1.000	0.895	0.604
Radius	1.111	1.139	1.117	1.000	0.674
Random	1.647	1.689	1.656	1.483	1.000

Table 6.4.2 *Compares the different methods of selection with the Direct coarsening scheme.*

rl1323	N.N	Greedy	Min Sum	Radius	Random
N.N	1.000	1.037	1.010	0.855	0.507
Greedy	0.964	1.000	0.974	0.824	0.489
Min Sum	0.990	1.027	1.000	0.846	0.502
Radius	1.170	1.214	1.182	1.000	0.593
Random	1.972	2.046	1.993	1.685	1.000

Table 6.4.3 *Compares the different methods of selection with the Direct coarsening scheme.*

u2319	N.N	Greedy	Min Sum	Radius	Random
N.N	1.000	1.007	1.006	0.888	0.611
Greedy	0.994	1.000	1.000	0.882	0.608
Min Sum	0.994	1.000	1.000	0.882	0.608
Radius	1.126	1.134	1.133	1.000	0.689
Random	1.635	1.646	1.645	1.452	1.000

Table 6.4.4 *Compares the different methods of selection with the Direct coarsening scheme.*

rl5915	N.N	Greedy	Min Sum	Radius	Random
N.N	1.000	1.089	1.025	0.780	0.347
Greedy	0.918	1.000	0.941	0.716	0.319
Min Sum	0.975	1.063	1.000	0.760	0.339
Radius	1.283	1.397	1.315	1.000	0.445
Random	2.880	3.138	2.953	2.246	1.000

Table 6.4.5 *Compares the different methods of selection with the Direct coarsening scheme.*

pla7393	N.N	Greedy	Min Sum	Radius	Random
N.N	1.000	1.073	1.041	0.862	0.306
Greedy	0.932	1.000	0.970	0.804	0.285
Min Sum	0.961	1.031	1.000	0.829	0.294
Radius	1.159	1.244	1.207	1.000	0.354
Random	3.272	3.511	3.406	2.822	1.000

Table 6.4.6 *Compares the different methods of selection with the Direct coarsening scheme.*

Average	N.N	Greedy	Min Sum	Radius	Random
N.N	1	1.0464	1.0176	0.857	0.4756
Greedy	0.9566	1	0.973	0.8208	0.4586
Min Sum	0.9828	1.0282	1	0.8424	0.4694
Radius	1.1698	1.2256	1.1908	1	0.551
Random	2.2812	2.406	2.3306	1.9376	1

Conclusion:

The Direct selection methods reproduce nearly the same results as the Recursive selection methods. But the difference between the different selection methods has changed somewhat. The Greedy method produced the best results in Direct coarsening as in Recursive coarsening, and all methods still produces better results than the Random selection. But the difference between the selection methods is less; N.N and Min Sum produce nearly identical results, especially in the smaller instances. The difference between Greedy and N.N is also less. Radius selection still produces the weakest result but the difference is less in the Direct scheme then in the Recursive scheme.

6.5 Comparing the Difference Between Direct and Recursive Coarsening

The last experiment for the different coarsening schemes is to compare the result from Direct coarsening with the Recursive coarsening selection. Since the last two experiments showed the different qualities for each selection type, this experiment compares the result from the Recursive method with its Direct counterpart.

Table 6.5.0 *Experiment settings:*

Parameters	***Value***
Search length	2N
Refinement	2.5-Opt
Temperature decrement factor	0.99
Chance of accepting move at start	30%
Selection Method	Varies
Method of extraction	BESTFIT

In this table we have calculated the normalized values by dividing the result from the Recursive scheme with the result of the Direct scheme. If the value is larger then 1, the Recursive scheme is longer then the Direct. If it is less then 1, the Recursive scheme is shorter.

Table 6.5.1 *Compares the difference between the Recursive and Direct coarsening scheme for each of the selection methods.*

	pr1002	rl1323	u2319	rl5915	pla7397	Average:
NN	0.991	1.006	1.007	1.034	1.010	1.010
Radius	1.071	1.084	1.066	1.122	1.110	1.091
Min sum	1.014	1.043	0.998	1.118	1.055	1.046
Greedy	1.015	1.016	1.004	1.095	1.050	1.036
Random	1.030	1.019	1.031	0.996	1.025	1.020

Conclusion:

Comparing the two different coarsening schemes it is obvious that the Direct coarsening performs better than the Recursive coarsening, for all the different selection methods.

This can only lead us to one conclusion. The Direct MLS is more powerful than the Recursive MLS. This means that the experiments of N. Bouhmala[18] where he shows us that the Direct scheme produces higher quality for Graph Partioning also holds for the Travelling Salesman Problem. An interesting observation is that this also holds for the Random selection method and we can therefore conclude that it is the coarsening scheme that produces the improvement effect. It can also be read from the table that the difference varies. Radius clearly produces a better result for Direct coarsening while the difference is very little for Nearest Neighbour.

6.6 Simulated Annealing Adaptations

As mentioned before S.A parameters needs to be tweaked before they work correctly. The best selection method and best scheme has now been identified; it is therefore now possible to start to adjust S.A so that it produces the best possible results with our best MLS.

6.6.1 The Length of the Search Path

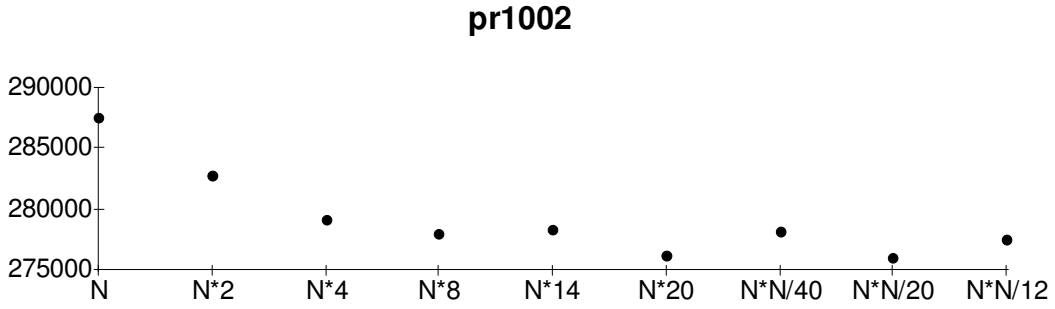
The length of the search path needs to be determined. There are two obvious choices for search path length, either a constant multiplier, or a variable dependent on the size of the map. The length of the search path is one of the main contributors to the running time for

S.A. The running time of S.A depends on the following factors *M*N*T*K*: Where *K* is the time for each perturbation, *T* is the number of temperature decreasing steps, *M*N* is the length of the search path which is a multiplier *M* of the Size of the problem *N*.
The search length is also dynamic because the Simulated Annealing does not lower the temperature before it reaches equilibrium. The solution reaches equilibrium if it has not change more than 2% in the last search step. Therefore, since we use this equilibrium condition, a constant value for *M* should be enough.

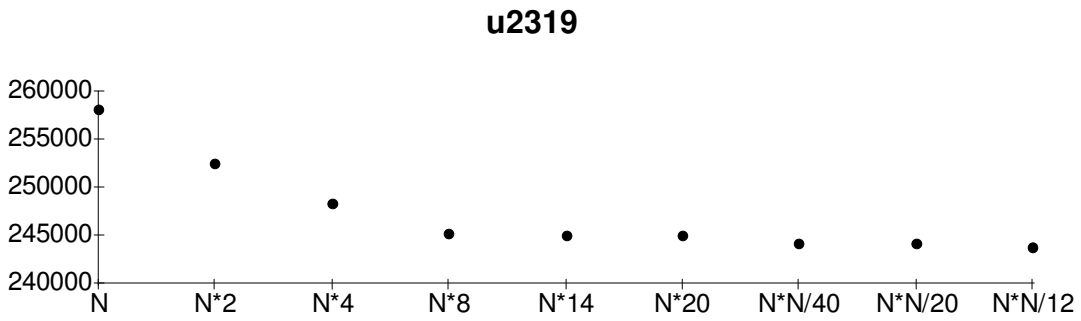
Table 6.6.1.0 *Experiment settings:*

Parameters	***Value***
Search length	Varies
Refinement	2.5-Opt
Temperature decrement factor	0.99
Chance of accepting move at start	30%
Selection Method	Greedy
Method of extraction	BESTFIT

The two graphs pr1002 and u2319 was selected for this experiment, the size is an important factor here, because testing for *M* close to *N* takes a long time, but the instances should sample the problem space well enough.



Plot 4.6.1.1 shows the length for different values of M on the graph pr1002.



Plot 4.6.1.2 shows the length for different values of M on the graph u2319.

Conclusion:

As one can see from the figures, a value of *M*=20 are well within the range where the solution for both instances stabilises. Then, if *M* is selected to be equal to 20, it should represent a sufficient value for *M* for most graphs to reach stability.

6.6.2 The Probability of Accepting Moves and the Temperature Decrement

The Probability of Accepting Moves

The start probability of accepting moves was proposed by N. Bouhmala [21] to be 30%. By experimenting with this value we found that the quality decreases slightly by reducing it to 20%. Reducing the value to 10% resulted in an increase of more then 5% difference in length compared with a value of 30%. By increasing the value to more then 30% there is a slightly gain but not enough compared to the increase in running time. Hence a value between 10% and 40% gave us the best results. The difference between 30% and 20% showed that 30% was a bit better than 20% but with an increase in running time. This time could be utilized in a different place to get better results. So a value of 25% for this variable is sufficient.

The Temperature Decrement

The temperature decrement is also an important factor in the total running time for Simulated Annealing. Usually in the literature this factor is set to be 0.95 [1][21]. Values from 0.90 to 0.99 were tested. A factor higher than 0.99 gives a higher quality but the running time increases very fast. Values lower then 0.90 results in a fast running time, but a lower quality solution. 0.95 is on average approximately 6% longer then 0.99 and 0.90 is about 8% longer than 0.95.
So on small graphs, sizes<10^4, a decrement factor of 0.99 can be used without problems. On sizes larger then 10^4 a decrement factor of 0.95 gives us a reasonable result.

Table 6.6.2.0 *Experiment settings:*

Parameters	***Value***
Search length	2N
Refinement	2.5-Opt
Temperature decrement factor	0.99/Varies
Chance of accepting move at start	Varies/25%
Selection Method	Greedy
Method of extraction	BESTFIT

The probability of accepting moves was performed with a temperature decrementing factor of 0.99. The temperature decrement factor was done with a probability of accepting moves at start of 25%.

6.6.3 Comparing the Different Methods of Permutation

Permutation is the most important part of the Simulated Annealing. It is the permutation that allows us to find the new solutions. Usually the permutation method is a K-Opt move, and we have focused on some of the best-known versions. We found it necessary to test several methods, since it would be interesting to see if there is a best possible permutation method that can be used with our MLS. We would also like to know if the quality discussion of different permutations discussed in [1], also holds for input produced by the MLS.

After some research into the field of local search strategies we decided to test the following permutations: Exchange, 2-Opt, 2.5-Opt and 3-Opt. All these permutations are described in detail in *Chapter 3.2,* and they can also be found presented in *Appendix 1*.

Table 6.6.3.0 *Experiment settings:*

Parameters	***Value***
Search depth	2N
Refinement	Varies
Temperature decrement factor	0.99
Chance of accepting move at start	30%
Merging Method	Greedy
Method of extraction	BESTFIT

As for the tables calculated for the selection experiments, these tables have normalized values. They are calculated by taking the resulting value from the row and divide it by the value in the column. The tables are therefore best read row by row. As for the last experiment 1 is equal, larger then 1 is longer and less then 1 is shorter.

Table 6.6.3.1 *Compares the different methods permutation*

pr1002	2-Opt	3-Opt	2.5-Opt	Exchange
2-Opt	1.000	1.202	1.214	0.868
3-Opt	0.832	1.000	1.010	0.722
2.5-Opt	0.823	0.990	1.000	0.715
Exchange	1.151	1.385	1.398	1.000

Table 6.6.3.2 *Compares the different methods permutation*

u2319	2-Opt	3-Opt	2.5-Opt	Exchange
2-Opt	1.000	1.165	1.166	0.936
3-Opt	0.858	1.000	1.001	0.804
2.5-Opt	0.858	0.999	1.000	0.803
Exchange	1.068	1.244	1.245	1.000

Table 6.6.3.3 *Compares the different methods permutation*

rl5915	2-Opt	3-Opt	2.5-Opt	Exchange
2-Opt	1.000	1.453	1.377	0.953
3-Opt	0.688	1.000	0.948	0.655
2.5-Opt	0.726	1.055	1.000	0.692
Exchange	1.050	1.526	1.446	1.000

Table 6.6.3.4 *Compares the different methods permutation*

pla7397	2-Opt	3-Opt	2.5-Opt	Exchange
2-Opt	1.000	1.417	1.348	0.903
3-Opt	0.706	1.000	0.951	0.637
2.5-Opt	0.742	1.051	1.000	0.670
Exchange	1.108	1.570	1.493	1.000

Table 6.6.3.5 *Compares the different methods permutation*

brd14051	2-Opt	3-Opt	2.5-Opt	Exchange
2-Opt	1.000	1.368	1.266	0.997
3-Opt	0.731	1.000	0.926	0.729
2.5-Opt	0.790	1.080	1.000	0.788
Exchange	1.003	1.372	1.270	1.000

Conclusion:

The test showed clearly that a weaker than 2-Opt method gave no gain in quality, and the use of such a method is therefore of no value in this case. The 2-Opt permutation produced acceptable quality for small instances, but for the larger instances the quality deteriorates quickly. The overall best permutation method was 3-Opt, but 2.5-Opt had approximately the same quality on the small graphs as 3-Opt. But since 2.5-Opt deteriorates in quality on larger instances, 3-Opt is better then 2.5-Opt.
The results found were comparable with the results from the comparison of different local search methods done in [1]. It is therefore possible to conclude that our scheme works as a regular TSP problem, also in the field of permutations. We can therefore get improved results by using a 3-Opt permutation when our result-tables are generated.

6.7 Reference Results

For every Travelling Salesman solver created it is important to provide reference-results. In that way other researchers can recreate or compare their results with ours.
As said in *Chapter 2.3* it was recommended in [1] to provide such results and use a problem set available for every interested researcher. We selected a set of instances from the TSPLIB packet, since instances from that packet are included in many studies of the Travelling Salesman Problem, and results are available from other researchers on the DIMACS [7] web site. The results provided is performed by using the best settings found in *Chapter 6*: The GREEDY selection method, combined with the Direct coarsening scheme, the extraction method BESTFIT and the different settings for Simulated Annealing. We used these settings on different instances to produce a reference table for a set of standard graphs.

The results provided can also be used by students who creates other coarsening schemes as reference for comparing their results; see *Appendix 4* for further research possibilities. In the table we present the results from Simulated Annealing, and for Simulated Annealing used as a greedy method. S.A is turned into this greedy version by setting the start temperature to 0, there by not allowing any uphill moves. The results are presented in percent excess longer than the optimal tour length. The reason for choosing to present also the result for the greedy S.A, is that it is a good tool for comparison when studying the effect of the S.A. Also included are the results from C.Walshaws Multilevel Lin-Kernighan (MLLK) [19]. The results can also be found in *Appendix 2* where a table presents the actual average length found for each instance.

Table 6.7.0 *Experiment settings:*

Parameters	***Value***
Search depth	20N
Refinement	3-Opt
Temperature decrement factor	0.95
Chance of accepting move at start	25%
Merging Method	Greedy
Method of extraction	BESTFIT

Table 6.7.1 *Presents the reference results*

Name	%ex S.A normal	%ex S.A Greedy	%ex MLLK
dsj1000	5.11	8.34	1.419
pr1002	5.33	7.02	2.093
u1060	5.29	6.29	1.703
vm1048	7.01	7.28	1.456
pcb1173	8.71	10.49	2.482
d1291	12.93	17.15	4.252
rl1304	7.25	9.89	1.282
rl1323	7.71	10.86	1.541
nrw1379	6.11	7.97	1.49
fl1400	3.77	5.65	1.168
u1432	6.7	9.62	2.108
fl1577	4.88	11.31	2.71
d1655	11.16	13.17	2.377
vm1748	8.33	8.39	1.958
u1817	12.82	15.56	3.5
rl1889	8.12	9.23	2.227
d2103	14.97	15.22	3.961
u2152	13.66	16.04	3.15
u2319	4.97	7.26	0.599
pr2392	7.86	9.49	2.489
pcb3038	10.25	11.36	1.944
fl3795	8.74	8.15	1.668
fnl4461	9.96	8.54	1.46
rl5915	17.21	14.05	2.152
rl5934	16.96	12.77	1.936
pla7393	12	9.47	1.724
usa13509	18.19	9	1.586
brd14051	18.73	8.66	
d15112	16.18	8.57	
d18512	19.44	9.36	

Comments:

The low quality in the larger instances is explained by the fact that the standard Simulated Annealing did not reach equilibrium on these instances within the constrains set by us. We can demonstrate this by comparing with the Greedy S.A where the quality for the larger instances is within the same result area as for the smaller instances. This effect is also visible in the experiments where the different selection methods were tested, where the Random selection produces a clearly weaker result. This is caused by the fact that S.A does not reach equilibrium for Random selection. The reason for this where that not enough work had been done on the larger instances before they reached the point of freezing. This where observed by lowering the temperature before the freezing test started. It was then possible observe that the quality improved for the larger graphs, but not for the smaller graphs. Since our goal was to produce a table where the same settings were used for all instances, we kept the above values.

Comparing the Results with Multi Level Linn Kernighan

It can be easily read from *table 6.7.1* that our results have weaker quality than the results produced by Multi Level Linn Kernighan (MLLK). Since MLLK is wrap around the Concorde TSP solver, we know that the refinement used in that scheme is perfected. By observing the differences in quality produced by MLLK, we can see that where MLLK produces weaker then average results. We also produce weaker then average results. This holds for all instances, so the results are possibly consistent.
But it looks like our results were weakened by not perfect implementation of the refinement method. This is one of the topics in the next chapter.

Chapter 7 Concluding the Experiments

7.1 An Overview of the Experiments

In *Chapter 5* the two developed coarsening schemes for the Travelling Salesman Problem, Recursive and Direct, were argued to hold the multilevel conditions from *Chapter 4.* Therefore they could be used to test the different aspects of the MLS.
Using the two different coarsening schemes proposed, we where able to test the different selection methods developed. In the experiments where the different selection methods was compared, the methods that were former tour construction heuristics was the champions, with the Greedy method as the one producing the highest quality.
By comparing the two different coarsening schemes for the different selection methods, the Direct coarsening method produced highest quality overall for all the different selection methods.
Since the Greedy method was the best selection method for the Direct coarsening scheme as well as for the Recursive coarsening scheme, the best combination for a coarsening scheme for the Travelling Salesman Problem according to my experiments would be: The combination of the Greedy selection method and the Direct coarsening scheme.

7.2 The Quality Gap Between Direct and Recursive coarsening

One of the more interesting discoveries was the difference in quality produced between the Recursive and the Direct coarsening scheme. In the graph partitioning paper of N. Bouhmala [18], it was shown that the Direct scheme produced higher quality than the Recursive scheme. By the comparison of the two different types of coarsening it was interesting to observe that this also holds for problems of the TSP type.
When we compared the two different coarsening schemes in *Chapter 6,* it is shown that the Direct coarsening scheme propagates clearly a better solution than the Recursive version. This also holds when the Random selection method is used.

Our observations can only lead to one conclusion: Somewhere in the Recursive scheme quality is lost. Since the refinement method should perform equally for the two coarsening schemes, we can assume that the quality is for some reason lost in the extension step. We can identify where the quality is lost in the Recursive scheme, by studying where the two different methods perform most of their work.
The main difference between the two methods is the number of cities added to the tour for each un-coarsening step. Since Recursive clusters are merged as binary trees, the current graph size is doubled for extension step. This leads to a problem in the last step where the graph is expanded from 50% to 100% graph size. In the Direct scheme this step size can be calculated by the formula (100/X) % where X is the number of cities in a Direct cluster. For an average value of 4, X would be ca 25% approximately for the Direct version. The step of 50% is probably too large for the extension method to cope with, and less quality is propagated from the previous refinement step. This makes the Recursive method top heavy compared with the Direct version that spreads the work out more evenly.

7.3 The Method of Extraction is Important

The extraction step is where the new start solution shall be created from the old refined solution. This step is important since it is here the quality found in the last refinement step shall be conserved and propagated to next step. In *Chapter 6* it was shown that it is not enough to just perform a random placement. Comparing the two different extraction methods in *Chapter 6*, demonstrates that the simple BEST FIT method transfers a higher quality from the lower level than the RANDOM method. The experiments performed on the extension step demonstrate that in this step, quality affecting the whole solution can be lost.

7.4 The Best Selection Method was Greedy

The difference between Greedy and the other selection methods was nearly the same for the different coarsening schemes. Greedy outperformed all the different selection schemes, on nearly all the different graphs. This result could be anticipated since the Greedy selection method was modelled after the Greedy tour construction heuristic. The Greedy tour construction algorithm is shown to function well in [1], and shown there to produce higher quality tours than the Nearest Neighbour tour construction heuristic. We anticipated that this result should hold also for a selection scheme based on the two heuristics. Since the methods based on tour construction heuristics produced the best results, it tells us that more advanced tour construction heuristic can possibly be a base for further studies into this field, see *Appendix 4*.

7.5 The Simulated Annealing Adaptation:

To find a universal setting for Simulated Annealing was problematic. The work performed to find the different settings lead us to the following conclusions: The best permutation method was 3-Opt. This is demonstrated in the experiment described in *Chapter 6.6.3*. The task of finding a universal setting of search length and temperature settings was more difficult. Especially a constant setting for M in the search length was hard to identify, but an acceptable value was found in *Chapter 6.6.1* and further used to produce the results for the reference table. The temperature decrement factor was also hard to set correctly. Decreasing it always lead to an increase in quality but also an increase in running time. Most authors recommended 0.95[1][21] and we chose therefore to use that value in the all graph reference table. In the comparing experiments the value was set to 0.99 to create higher resolution, to be able to discern the trends better. The correct settings for S.A were difficult to identify, and as discussed before, equilibrium was not reached with our version of Simulated Annealing for the larger graphs. This fact can be explained by different factors but mainly it looks like the solution did not reach stability before it was stopped by the freezing point test. We can therefore conclude that not enough work was performed by the refinement method on the larger instances for it to be able to reach equilibrium on them.

Chapter 8 Concluding the Thesis

In this paper two different schemes for the multilevel structure has been developed, tested and compared with each other. A set of selection methods has been developed and tested for the two MLS developed, and the best combination was found. The Simulated Annealing was adapted to work in the ML environment and different settings for S.A, was tested until a universal best was found. The best settings were used to produce a reference table for further studies.
We found that the ML-Scheme with S.A did not produce a result with the quality we hoped for in the beginning of this project. The reasons were discovered and possible ways to work around this problem is proposed in *Appendix 4 further research possibilities.*

References:

[1] The Travelling Salesman Problem: A case study in local optimization by
David S. Johnson and Lyle A. McGeoch 1995

[2] Introduction to ALGORITHMS
MIT Press
Thomas H. Cormen, Charles E. Leiserson and Ronald L. Rivest 1998

[3] Web source:
http://mathworld.wolfram.com/IcosianGame.html
Eric W. Weisstein (Last Update 1999)

[4] The history of combinatorial optimization (till 1960)
Alexander Schrivjer

[5] Web Source:
http://www.math.princeton.edu/tsp/milestone.html
William J. Cook (bico@math.princeton.edu) (Last Update March 2002)

[6] Ulysses 2000: In Search of Optimal Solutions to Hard Combinatorial Problems,
Technical Report, New York University Stern School of Business.
M. Grötschel and M. Padberg, (1993).

[7] Web Source:
http://www.research.att.com/~dsj/chtsp/
David S. Johnson (Last Update 19 June 2002)

[8] Web Source:
http://mie.eng.wayne.edu/faculty/chelst/informs/case-studies/metelco.htm
Anonymous; Industrial and Manufacturing Department - Wayne State
University (Last Update 18 August 1997)

[9] Models of computation, Exploring the Power of Computing
Addison Wesley Publishing Company
John E. Savage 1998

[10] Web Source:
http://www.ncbi.nlm.nih.gov/genome/rhmap/
Richa Agarwala (Last Update Not Posted)

[11] Fuel-Saving Strategies for Dual Spacecraft Interferometry Missions
Journal of the Astronatical Sciences Volume 49, No. 3
Christopher A. Bailey, Timothy W. McLain, Randal W. Beard

[12] Powering the Last Mile
Alpha Technologies white papers
Thomas H. Sloane Kurt Gutierrez 1997

[13] Web Source:
http://www.math.princeton.edu/tsp/apps/dna.html
William J. Cook (bico@math.princeton.edu) (Last Update December 5, 2001)

[14] Asymptotic Experimental Analysis for the Held-Karp Traveling Salesman Bound
proceedings of the 7th annual ACM-SIAM Symposium on Discrete Algorithms(341-350). D. S. Johnson L. A. McGeoch E. E. Rothberg 1996

[15] Euclidean TSP is NP-complete.
Theoretical Computer Science, 4:237-244,
C. H. Papadimitriou 1977

[16] The travelling-salesman problem,
Operations Research 4**,** 61-75
M.M. Flood 1956

[17] Optimization by Simulated Annealing,
Scince, 220, No. 4598. 498-516,
S. Kirkpatrick, C. Gelatt, Jr., and M. Vecchi 1983

[18] An experimental comparison of Different graph coarsening schemes
Noureddine Bouhmala

[19] A Multilevel approach to the travelling salesman problem
Mathematics research report 00/IM/63
Chris Walshaw 2000

[20] Calculus a complete course Third Edition
Addison Wesley Publishing Company
Robert A. Adams 1995

[21] Correspondence:
Simulated Annealing
Noureddine Bouhmala 2001

[22] Web Source:
http://www.math.princeton.edu/tsp/concorde.html
William J. Cook (bico@math.princeton.edu) (Last Update 3 July 2001.)

[23] Computer Simulation Methods,
Addison-Wesley Publishing Company
H. Gould, J. Tobochnik 1988.

[24] Introduction to the theory of computation
PWS publishing company
Michael Sipser 1997

[25] Web Source:
http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/
Gerhard Reinelt (Last Update 6 December 2001)

[26] Web Source:
http://www.research.att.com/~dsj/chtsp/download.html
Anonymous (Last Update 28 May 2002)

Reference books used in the programming:

[27] Mastering Algorithms with C
O'Reilly & Associates, Inc.
Kyle Loudon 1999

[28] C BY EXAMPLE
Que Corporation
Greg Perry 2000

References used in the preface

[29] Web Source
http://www.math.princeton.edu/tsp/travelling.html
William J. Cook (bico@math.princeton.edu) (Last Update 22 March 2002.)

[30] Oxford Advanced Learner's Dictionary of Current English *Fifth Edition*
Oxford University Press
Jonathan Crowther, Kathryn Kavanagh, Michael Ashby 1995

[31] http://www.math.princeton.edu/tsp/history/d15112.html
William J. Cook (bico@math.princeton.edu) (Last Update 27 March 2002.)

Appendix 1 Methods Used in Detail

In this appendix some of methods used in our implementation is presented in pseudocode. The object is to give the reader a deeper understanding of some important structures, and to complement the discussion in *Chapter 5*. For each method, there is presented a short analysis of special elements and a worst-case analysis.

Distance calculating between two cities (clusters)

To calculate the distance from city A to city B, the euclidian distance formula for points in the 2D plane is used:

***Definition*: The euclidian distance formula**

For the 2 points in the plane (x_1, y_1) and (x_2, y_2) the distance between them equals $\sqrt{((x_2-x_1)^2 + (y_2-y_1)^2)}$ [20]

From this formula it is possible to derive a method for calculating the distance between two cities in constant time. Since the input type is symmetric, the order of the input values A and B is not important.

Name:	**DIST CALC**
Input:	Two cities A, B.
Output:	The distance between the two cities.
1 i ←(B(x)-A(x)) ² 2 j ←(B(y)-A(y)) ² 3 **return root** (i+j)	

Running time analysis:

All operations are constant time operations; this leads to a constant running time.

Length of Tour calculation

The length of a travelling salesman tour is the sum of the length of every edge in the tour se *Chapter 1.1*. To calculate the tour length, the algorithm walks the travelling salesman tour. For each new edge the algorithm traverses, the length of the edge is added to the sum of the tour length.

Name:	**TOUR LENGTH**
Input:	A TOUR
Output:	The total length of the TOUR

1 **while** all edges in TOUR are not yet visited

1.1 Get next unvisited edge form TOUR

1.2.1 A← Start edge city

1.2.2 B← End edge city

1.3 dist←dist + **Dist Calc** (A, B)

2 **return** dist

The direction and the starting point of the traversal is not important since the instance is symmetric and euclidian, and as long as all the edges are visited this method will produce the length of the tour.

Running time analysis:

Since the distance calculation is constant, this operation can be done time order O(N) where N is the number of cities in the TOUR.

Nearest Neighbour coarsening

For each of the selection methods, we have chosen presented them in the Recursive structure. By comparing the methods with the Direct structure presented in *Chapter 5* it is easily seen how these selection methods can be converted to the Direct scheme.

Name:	**Nearest Neighbour**
Input:	MAP
Output:	A coarsened level

1. **While** still unmerged clusters in MAP
    1.1 Select a random cluster A from MAP
    1.2 **for** all unmerged cities in MAP
        1.2.1 Select B the nearest cluster to A
    1.3 **Merge**(A, B)
    1.4 Set AB as merged

Comments on operations:

Line 1.2.1: This can be done in two different ways, either with a pre-generated distance matrix or by finding the best distance for each merging. This does not affect the order of the running time because the creating of the distance matrix takes an order $O(N^2)$ running time. In practice the running time is in fact shorter for the second option since N decreases by two for every iteration.

The running time of the algorithm

The loop in *line 1* is executed N/2 times, the loop in *line 1.2* is executed N-I times for each iteration of N where I increase with 2 for each time the loop is executed. The merging operation is constant. So the running time is of order $O(N^2)$, where N is the number of cities.

Greedy coarsening

Name:	**GREEDY**
Input:	MAP
Output:	A coarsened level

1.**for** all cities in MAP

    1.1 Generate entries in the distance matrix

2.**for** all cities in MAP

    2.1 Locate the shortest edge

    2.2 Add it to the shortest edge list

3. **While** still unmerged cities in MAP

    3.1 **for** all unmerged cities in map

        3.1.1 locate the shortest edge in the shortest edge list.

    3.2 Let A and B be the vertexes connected to the edge

    3.3 **MERGE**(A ,B)

    3.4 Update the shortest edge list

Comments:

Line 1: Entries in the distance matrix are found for each city by calculating the distance to all other cities on the map. Since the problem is symmetric only the upper triangular part of the heuristic needs to be calculated.

Line 2: This creates a list where the shortest edge for each city/cluster is stored.

Line 3.4: The update step is to locate all of the shortest edges with one of the removed cities/clusters as endpoints.

Running time analysis:

Generating distance matrix in *line 1* is done in $O(N^2)$ time. Creating the Shortest edge List is done in $O(N^2)$ time. The loop in *line 3* is executed N/2 times. The Loop in *line 3.1* is executed for N-I times where I increase with two for each execution where N is the number of cities. *Line 3.4* has a running time in K*N where K is the number of edges to be updated and K is usually much smaller then N but it is possible to have a value close to N.
This gives us a total running time of $O(N^2*K)$. It is also possible, by using a heap structure for each node, to get a running time of $O(N^2 \log N)$. But it is not necessarily worth the extra implementation work since K is usually much smaller then N.

Max Min coarsening

Name:	**MAXMIN**
Input:	MAP
Output:	A coarsened level

1.**for** all cities in MAP

1.1 Generate entries in the distance matrix

2.**for** all cities in MAP

2.1 Locate the shortest edge

2.2 Add it to the shortest edge list

3. **While** still unmerged cities in MAP

3.1 **for** all unmerged cities in MAP

3.1.1 locate the longest edge in the shortest edge list.

3.2 Let A and B be the vertexes connected to the edge

3.3 **MERGE**(A,B)

3.4 Update the shortest edge list

Comments:

Line 1: Entries in the distance matrix are found for each city by calculating the distance to all other cities on the map. Since the problem is symmetric only the upper triangular part of the heuristic need to be calculated.

Line 2: This creates a list where the shortest edge for each city/cluster is stored.

Line 3.4: The update step is to locate all shortest edges where the endpoints was one of the removed cities/clusters.

Running time analysis:

Generating distance matrix in *line 1* is done in $O(N^2)$ time. Creating the Shortest edge List is done in $O(N^2)$ time. The loop in *line 3* is executed N/2 times where N is the number of cities. The Loop in *line 3.1* is executed for N-I times where I increase with two for each execution. *Line 3.4* has a running time in K*N where K is the number of edges to be updated and K is usually much smaller then N but it is possible to have a value close to N. This gives us a total running time of $O(N^2*K)$.

Square coarsening

Name:	**SQUARE**
Input:	MAP
Output:	A coarsened level

1 calculate *h*

2 generate the buckets for bucket sort

3 **for** all cities in MAP

    3.1 sort the cities in to the buckets using bucket sort

4 **for** each bucket

    4.1 Merge with the closes vacant cluster within the bucket or one of the adjacent buckets.

Comments:

Line 2: The number of buckets needs to be calculated to be equal to the number squares.

Line 3.1: By comparing the position of the city/cluster it is straightforward to calculate which bucket the city/cluster is placed within.

Line 4.1: This step merges the cities/clusters within each bucket, by a merging method. The same method as C.Walshaw selecting the closest cluster/city where used.

Running time analysis

In *line 1 h* is calculated in constant time. The calculation of the size of the bucket is done in constant time. The loop in *line 3* is executed N times.
The running time of *line 4* and *4.1* M*L where M is the number of buckets and L is the running time of merging the cities within each bucket giving us a worst case of $O(N^2)$.

Radius coarsening

Name:	**RADIUS**
Input:	MAP
Output:	A coarsened level

1 Calculate *h*

1.2 $r \leftarrow h/2$

2 **while** still unmerged cities in MAP

2.1 randomly select a city A from MAP

2.2 **while** no merge candidate had been found

2.2.1 pick a new B from MAP

2.2.2 **if Dist Calc** (A, B)<*r*

2.2.2.1 merge candidate found

2.3 **MERGE**(A,B)

2.4 remove A and B from MAP

2.5 **if** no merge was found remove A from MAP

Comments:

Line 1: *h* is calculated as explained for Square.

Line 2.5: If there was no city/cluster within the radius the city/cluster is removed and cannot be merged before *h* is increased, in the next level of merging.

Running time analysis:

In *line 1 h* can be calculated in constant time. The loop in *line 2* is executed no more then N/2 times. The Loop in *line 2.2* is executed less then N
This gives us a worst case running time of $O(N^2)$.

Random coarsening

Name:	**RANDOM**
Input:	MAP
Output:	A coarsened level

1 **while** still unmerged cities in MAP

1.2 Select a random cluster A from MAP

1.3 Select a random cluster B from MAP

1.4 **MERGE**(A, B).

1.5 Remove A and B from MAP

Running Time analysis

Line 1 is executed N/2 times and is the only line in the heuristic without constant running time this gives us a total running time of O(N).

RANDOM Extension

Name:	**RANDOM**
Input:	TOUR
Output:	Extended TOUR

1 **for** each cluster in the tour

1.2 **EXTRACT** (AB)

1.3 i ←**RANDOM** (1,2)

1.4 **if** i ==1

1.4.1 add the new edge A-B to TOUR where the cluster A was

1.5 **else**

1.5.1 add the new edge B-A to the TOUR where the cluster A was

Running Time:

The loop in Line 1 is executed N times where N is the Number of Clusters before the extraction. All other lines are constant time operations. So the running time is O(N).

Best Fit Extension

Name:	**BEST FIT**
Input:	TOUR
Output:	Extended TOUR

1 **for** each cluster in the tour

1.2 extract the cluster in to two clusters A and B

1.3 x← **Dist Calc** (X, A)+ **Dist Calc** (A, B)

+ **Dist Calc** (B, Y)

1.4 y← **Dist Calc** (X, B)+ **Dist Calc** (B, A)

+ **Dist Calc** (A, Y)

1.5 **if** x<=y

1.5.1 add the new edge A-B to TOUR where the cluster A was

1.6 **else**

1.6.1 add the new edge B-A to the TOUR where the cluster A was

Comments:

Line 1.3 and line 1.4 calculates the distance of the two line segments.

Line 1.5→Line 1.6.1 compares the length of the two line segments and inserts the shortest configuration to the tour.

Running time:

The loop in *line 1* is executed N times where N is the Number of Clusters before the extraction. All other lines are constant time operations. So the running time is O(N). But the constant factor in BEST FIT is larger than RANDOM.

Exchange permute

Name:	**EXCHANGE PERMUTATION**
Input:	TOUR
Output:	The changed TOUR, and new length of tour
1 randomly select a city A from TOUR 2 remove A from TOUR 3 add A to TOUR in the position before the next cluster 4 Recalculate the length of the tour	

2.5-Opt

Name:	**2.5-OPT PERMUTATION**
Input:	TOUR
Output:	The changed TOUR, and new length of tour
1 randomly select two cities A and B 2 remove A from the tour 3 add A in the position before B in the tour 4 recalculate the new tour length	

2-Opt permute

Name:	**2-OPT PERMUTATION**
Input:	TOUR
Output:	The changed TOUR, and new length of tour
1 randomly select two clusters A and C from TOUR 2 remove the edge from A to cluster B 3 remove the edge from C to cluster D 4 Add a new edge from A to C 5 Add a new edge from B to D 6 Recalculate the length of the new tour	

3.3.2.5 3-Opt permute

Name:	**3-OPT PERMUTATION**
Input:	TOUR
Output:	The changed TOUR, and new length of tour
1 Randomly select 3 different cities from the tour A, C and E 2 Remove edges from A to B, C to D and E to F 3 Find the best possible way to reconnect TOUR	

Comments:

Line 3: Because there are 6 different ways to reconnect the graph legally after 3 edges are cut, the move gain for each of those has be calculated and compared to find the best solution.

Appendix 2 Result table

Name	S.A Normal(1)	S.A Greedy(2)	Optimal length
dsj1000	19613159	20216530.52	18659688
pr1002	272862.73	277220.67	259045
u1060	235944.97	238191.4809	224094
vm1048	256078.24	256719.3151	239297
pcb1173	61848.223	62860.5487	56892
d1291	57371.282	59511.17548	50801
rl1304	271277.96	277965.7688	252948
rl1323	291025.98	299537.9101	270199
nrw1379	60100.395	61150.88654	56638
fl1400	20885.55	21264.89033	20127
u1432	163213.2	167692.594	152970
fl1577	23333.946	24766.09718	22249
d1655	69061.857	70309.77203	62128
vm1748	364580.26	364802.4836	336556
u1817	64533.316	66102.95033	57201
rl1889	342234.95	345766.3542	316536
d2103	92496.5	92691.6606	80450
u2152	73030.23	74557.65995	64253
u2319	245902.62	251266.8863	234256
pr2392	407729.38	413914.7589	378032
pcb3038	151807.13	153334.2312	137694
fl3795	31286.647	31117.32661	28772
fnl4461	200743.01	198161.0175	182566
rl5915	662873.8	644962.8326	565530
rl5934	650353.49	627027.908	556045
pla7393	26062569.6	25473219.03	23269728
usa13509	23617121.55	21780588.88	19982859
brd14051*	557269.7909	510009.7391	469375
d15112*	1827542.619	1707873.612	1573040
d18512*	770659.0438	705598.9887	645230

Appendix 3 TSPLIB

This table holds the different instances used from the TSPLIB package.
The table shows the name of the instance, the size of the instance and optimal length.
The instances marked with * is within +/- 0.05% of optimal length.

Name	Size	Optimal
dsj1000	1000	18659688
pr1002	1002	259045
u1060	1060	224094
vm1048	1048	239297
pcb1173	1173	56892
d1291	1291	50801
rl1304	1304	252948
rl1323	1323	270199
nrw1379	1379	56638
fl1400	1400	20127
u1432	1432	152970
fl1577	1577	22249
d1655	1655	62128
vm1748	1748	336556
u1817	1817	57201
rl1889	1889	316536
d2103	2103	80450
u2152	2152	64253
u2319	2319	234256
pr2392	2392	378032
pcb3038	3038	137694
fl3795	3795	28772
fnl4461	4461	182566
rl5915	5915	565530
rl5934	5934	556045
pla7393	7393	23269728
usa13509	13509	19982859
brd14051*	14051	469375
d15112*	15112	1573040
d18512*	18512	645230

Appendix 4 Further Research Possibilities

Other Refinement Methods

Further studies should investigate the impact of different refinement methods with the best scheme found in this study. An interesting possibility is to develop a ML interface for the Direct Multilevel scheme to the perfected Linn-Kernighan program Concorde [22] developed by a Princeton research group.
Other interesting possibilities are to use the ML-Scheme in a study of more "exotic" refinement algorithms for example the genetic refinement mentioned in *Chapter 3.4.*

The Extension Step and The coarsening Step

From the comparing of the different MLS developed, is it obvious that quality most likely is lost in the extension step. A further research possibility is to improve the method of extraction to expect more the way that particular edge should be inserted. Since the best selection methods were adaptations of different tour construction heuristics, another possible research subject could be to look for other tour construction heuristics that are adaptable to the MLS.

Another ML-Structure

Another possibility is to change the way we look at clusters and thereby remove the problem we have seen with the extension. Since we loose quality when we increase the graph size by splitting clusters and thereby adding new edges to the tour, one possibility is to create a Multilevel scheme that works on a fully extended graph. Where it instead of merging cities into clusters, locks city connections in to place. Then the clusters would be organized as rigid line segments in the tour, where the length of the line segment is known. This solution let the refinement work with a full-length tour but holds the conditions for the Multilevel scheme, since it coarsens the problem by just letting a few edges be free for change. This solves the extension problem because the extension for such a problem would be to free edges instead of adding new edges to the tour, and no quality of the previous solution is lost because of the insertion of new edges.
Therefore it should be interesting to test the selection methods we have developed for such a scheme.

Appendix 5 List of Abbreviations

CW	Clark- Wrigth
LK	Lin-Kernighan
MLLK	Multilevel Lin-Kernighan
MLS	Multilevel Scheme
NP	Non-deterministic Polynomial Time; See *definition 2.1.2*
P	Polynomial Time; See *definition 2.1.1*
S.A	Simulated Annealing
TSP	Travelling Salesman Problem
TS	Tabu search
N.N	Nearest Neighbour

Appendix 6 Abstract

The Travelling Salesman Problem is one of the intriguing problems from the world of computational complexity. In this paper a new structure for solving computational hard problems; the Multilevel scheme, is introduced.
By using the Multilevel scheme combined with the large resource of existing studies of the Travelling Salesman Problem, we whished to see if the Travelling Salesman problem could be solved by using the Multilevel scheme. By developing and studying tools, we will try to refine the eventual quality produced by the Multilevel scheme.
The Multilevel scheme is a technique, which limits the problem size in such away that a refinement method can be used on a smaller problem. The refinement on the smaller problem can then be used to solve the original problem with higher quality.
We found that it was possible to use the Multilevel scheme on the Travelling Salesman Problem. We also found and studied tools that can be used in further studies; we identified also directions and ideas in areas where new techniques for improving the results with the Multilevel scheme can be found. By comparing different Multilevel scheme structures we found that they differed in quality, our research can therefore also be interesting for other problems where the Multilevel scheme can be used.