Using improvement location and improvement preference to create meta-heuristics

Edward Parkinson

Doctor of Philosophy

City University

Department of Computing

June 2004

Contents

Tables and Figures

Acknowledgements

Many thanks to my supervisors Andrew Tuson and Peter Smith. Andrew, your suggestions and comments have been invaluable, I have lost track of the number to times you have read drafts for me. Peter thanks for the feedback and especially your suggestion of having a go at implementing some of the algorithms, it give me a real feel for the problems.

Thanks to the staff who look after Broomfield Park in north London. Most of the ideas in the thesis originated while I was in the park.

Thanks to Marcus Andrews, Panos Dafas, Barry Kwam, and Claire Thie for reading through thesis drafts and giving me feedback. You spotted many mistakes and your comments help me understand what was and what was not being understood.

Also thanks to the many people who dragged me away from my thesis once in a while and I would not have got it written without you.

Eddy Parkinson (eddyparkinson@yahoo.co.uk)

Abstract

Local search algorithms are used to produce solutions to combinatorial logistics problems such as the vehicle routing problem. Where the algorithms aim to minimise the number of vehicles needed to deliver goods.

Local search creates a first solution and that solution is the best until the heuristic finds a better one, hence it is argued that all existing local search algorithms make a series of improvements. Different local search algorithms locate these improvements with different improvement location mechanisms. Local Search also gives preference to using some improvements rather than others. To make these preference choices different local search algorithms use different improvement preference mechanisms.

Current local search algorithms intertwine improvement location and improvement preference mechanisms, making it difficult to identify if they are both having a positive impact and the scale of the impact. The thesis hypothesis states "Distinguishing between improvement location and improvement preference is feasible and useful when creating local search algorithms". Distinguishing between improvement location and preference produced results that suggest how local search methods in current use can be improved.

The experiments compare methods of locating improvements and improvement preference. They use both whole non-improving solutions and partial solution changes, similar to Lin-Kernighan, to locate improvements. They also compare giving preference to small improvements or large improvements. The comparisons are done using the Solomon vehicle routing problem benchmarks. Using partial solution changes to locate improvements and giving preference to small improvements typically gave the highest quality solutions at the cost of doubling execution time. The results also show which methods of location and preference do well when only a small amount of execution time is available.

Keywords: optimisation, search-control, local-search, meta-heuristics, vehicle-routing-problem, Lin-Kernighan, ant-colonies, genetic-algorithms, simulated-annealing, tabu-search.

Abbreviations and Terms

AC	Ant-Colonies – term used to describe local search methods based on ant pheromones
AI	Artificial Intelligence - AI can be seen as an attempt to model aspects of human thought on computers
Compound Move	When one move is combined with another. Instead of a large number of changes being made in a single move, two or more smaller moves are used to achieve the same result.
Construction Heuristics	They create new solutions from scratch. A simple construction method for the TSP is to pick a customer at random then keep travelling to the nearest unvisited customer until all the customers have been visited.
EA	Evolutionary Algorithm - used to describe evolution based local search.
Global Optimum	The best quality solution to a problem.
Greedy Heuristic	Has a short-term 'greedy' strategy of picking what looks to be the best element or change available.
Hill Climber	Takes a solution to a problem and makes a series of improvements to it. No guarantees are make about how good the end solution will be, as it is not designed to check for all possible improvements.
Improvement Location	Takes one of more solutions and uses them to create a new solution better than the current best.
Improvement Preference	Giving preference to using one solution rather than another to locate improvements.
Iterated Local Search	Takes a solution produced by a local search method, then makes the solution worse, then re-runs the local search heuristic
Lin-Kernighan	Uses partial solution changes to guide its search for improvements.
Local Optimum	A solution that can not be improved by making small changes to it.
Local Search	Takes an existing solution and uses educated guesses to find ways of improving the solution.
Lower Bound	Feasible solutions can not be better than the lower bound. The lower bound is normally slightly less than the best solution (when minimising).
Meta-heuristic	This is general heuristic that can be applied to more than one problem. A meta-heuristic is a kind of high level framework that can be use to create an algorithm for a specific problem.

NP	Non-deterministic Polynomial - Roughly speaking this means the time needed to find the best possible solution to a problem becomes excessively large with larger problem sizes.
Opt (k-opt)	Repeatedly making 2-opt improvements to a solution typically produces a sub-optimal solution, this solution is classed as 2-optimal, hence the term 2-opt.
OR	Operational Research – is taking a real-world problem and creating an optimised solution for it.
Partial Change	Like a whole move except only some of the changes are implemented.
SA	Simulated Annealing - takes a solution and mostly makes improvements to it, although occasionally it also makes the solution worse, it was inspired by processes in statistical mechanics, in physics.
SAHC	Steepest Ascent Hill Climber – takes a solution and implements the best small change it can find and then repeats the process.
Threshold methods	Takes a solution and makes a small change to it, all improving changes are accepted and the threshold for accepting non-improving changes is gradually raised. This means they reject more and more non-improving changes as the search progresses.
TS	Tabu Search – takes a solution and makes small improving changes to it. When no more exist it allows non-improving changes, but only if they have not recently been undone or redone, this prevents cycling.
TSP	Travelling Salesman Problem - a sales-person needs to visit a number of customers. The aim is to try and work out the shortest route the salesperson can take in order to visit each customer once.
VRP	Vehicle Routing Problem – the problem of delivering goods to many different customers using a fleet of vehicles.
VRPTW	Vehicle Routing Problem with Time Windows – like the VRP but with restrictions on when the goods can be delivered to customers.
Whole Move	Taking a complete and valid solution and changing it to create a new complete and valid solution.

1) Introduction and overview

Companies use local search algorithms to reduce costs. They are used on problems such as organising the delivery of goods using a fleet of vehicles [Rochat 94], production line scheduling [Dorn 95]. Because of this, improving how these algorithms work means companies and organisation that use them can reduce costs. Although there are many local search algorithms [Reeves 93] and companies want to know which ones suit their optimisation problem.

Local search algorithms take an existing solution and use educated guesses to find ways of improving the solutions. See the background chapter, Chapter 2, for a description of local search.

There appears to be a constant stream of new local search algorithms described in the literature, see [Corne 99] "New Ideas in Optimization" for some of the more recent ones. The problem is, it is hard for companies to work out which of these local search algorithms is suitable for their optimisation problem. This problem is summed up by [Papadimitriou 82]:

> "...the design of effective local search algorithms has been, and remains, very much an art."

To help deal with this problem, a method of classification is proposed and tested that allows two of the mechanisms of local search algorithms to be separated. This means the impact of each of these mechanisms on different problems can be individually measured.

The two local search mechanisms are improvement location and preference. Local search algorithms make a series of improvements. Current local search algorithms do not distinguish between the mechanisms that locate these improvements and the mechanisms that give preference to using some of these improvements rather than others. Distinguishing between the two means the value of each mechanism can be assessed.

The thesis makes both general theoretical claims and describes practical experiments that test these claims. The main theoretical claim is that it is worth distinguishing between improvement location and preference mechanisms. This theoretical claim is investigated using practical experiments. The practical experiments evaluate the usefulness and feasibility of distinguishing between the mechanisms using Vehicle Routing Problem (VRP). The VRP is the problem of delivering goods to customers using a fleet of vehicles. The experiments test the performance of difference improvement location and preference mechanisms on variations of the delivery problem and evaluate which mechanisms work best on which problem variation of the problem.

The experiments test how long the different mechanisms take to create an acceptable solution, this is because with large problems it can take a long while to create an acceptable solution and so mechanisms that reduce the length of time needed are valuable. The experiments also examine problems with slack and tight delivery times, this is done to test how well the mechanisms handle difference styles of delivery problem. The results show how improvement location and preference

mechanisms can be matched to the demands of different VRP problem types and reveal refinements that can be made to the mechanisms.

The results show individual improvement location and preference mechanisms are suited to some VRP style problems and not others. For example giving preference to using large improvements in solution quality produced its best solutions in a few seconds. This is useful when solutions need to be created quickly but when several minutes or hours are available other methods tended to perform better. The results also show how refinements made to a commonly used preference mechanism could simultaneously improve solution quality and execution time, see section 7.4.1.

1.1) Definitions and scope

See chapter 4 and the Lin-Kernighan section of chapter 3 for more detailed descriptions of each of these: -

Local search takes an existing solution and makes a series of improvements to it. Only local search methods are examined in depth by this thesis.

Improvement location uses one or more previous solutions to find an improvement. All local search methods use previous solutions to help locate improvements. This thesis compares methods of locating improvements. **Note:** An improvement location mechanism is a mechanism that locates a solution that is better than the best solution found so far, this is fundamental to the arguments made in the thesis.

Improvement preference gives preference to using one solution improvement rather than another to locate further improvements. This does not include improvements that are always rejected, e.g. with the aim of speeding up the optimisation process. This thesis compares improvement preference choices.

Whole solution is a solution that is both complete and valid. Complete in the sense that no parts of it are missing and valid in the sense that it looks like it can be implemented.

Whole move is when a complete and valid solution is changed to create a new complete and valid solution. The move is whole because both solutions are 'whole', i.e. complete and valid. It is called a move because it is the process of moving from one solution to another.

Partial change (partial solution change) is exactly like a whole move except only some of the changes are implemented. Applying two or more partial changes to a whole solution produces a new whole solution. Using more than two partial changes to 'move' from one whole solution to another means the two whole solutions have more differences. The partial change concept is based on Lin-Kernighan [Lin 73].

Partial solution is produced when a partial change is applied to a whole solution. A partial solution is invalid and incomplete. Any partial solution can be converted into a whole solution using a single partial change, this means a partial solution is never more than one step away from a whole solution.

1.2) Key points and overview of thesis.

Motivation
Many logistics problems exist, for example, organising the delivery of goods using a fleet of vehicles. Companies use local search algorithms to create solutions to such logistics problems. The better the local search algorithm, the better the solution, a better solution means lower costs in terms of both labour and resources.

Research objectives
The aim of the research was to identify which local search methods work well with which logistics problems. To achieve this the research identifies improvement location mechanisms and improvement preference mechanisms that exist in local search methods. These local search methods and the improvement location and preference that they use are described in depth in Chapter 4. Chapter 7 describes various improvement location and improvement preference mechanisms and shows which logistics problems they worked best with.

Summary of objectives and results

- Show it is feasible to distinguish between improvement location and improvement preference using an existing example logistics problem.

The results show it is feasible to distinguish between improvement location and preference. See Chapter 5 for details and pseudo code.

- Identify advantages and disadvantages of using different improvement preference choices.

Using a preference choice that is the opposite of what is currently recommended produced the best quality solutions, see section 7.4.2. This looks to be at the cost of doubling execution time, see section 7.4.3. Changing the preference allowed execution time to be decreased, for problems that require a shorter execution time, see sections 7.2 and 7.4.2.

- Identify advantages and disadvantages of using different improvement location methods.

The results showed which improvement location methods were suited to which VRP problems, see sections 7.2 & 7.3. Adjustments to the location method also allowed execution time to be controlled. This looks to be valuable for fine tuning the execution time and solution quality, see section 7.4.1.

- Show if Lin-Kernighan [Lin 73] style improvement location can successfully be used on the Vehicle Routing Problem.

Lin-Kernighan [Lin 73] style improvement location typically created the best or near best quality solutions, see section 7.3. It tended to perform best when customers were clustered. Although Lin-Kernighan [Lin 73] style improvement location was typically less than 2% better than using whole moves to locate improvements.

It proved difficult to model customer delivery times and easy to model distance with Lin-Kernighan style improvement location. The Lin-Kernighan style methods that were guided using customer delivery times typically performed badly, see section 7.5.3. The Lin-Kernighan style methods that where guided by distance alone tended to perform best.

General contribution to knowledge

Identifying which methods work well on which logistics problems aids algorithm design. Distinguishing between improvement location and improvement preference improves the identification of which methods work with which problems. See Chapter 4 for a description of how previous solutions are used to locate improvements and how preference is given to using some improvements rather than others.

Practical Contribution to knowledge

Reducing the cost of delivering goods is a practical problem. Distinguishing between improvement location and improvement preference revealed methods that reduced costs more than the methods currently in use. The results show that different methods of improvement location and preference are suited to different VRP problem styles, see results chapter.

Feasibility

Distinguishing between improvement location and improvement preference is feasible. The experiments show how several methods of improvement location and preference can be compared. The experiments chapter describes how the comparisons were done.

Gap in knowledge

Current local search methods do not distinguish between improvement location and improvement preference. They treat improvement location and preference as one. Chapter 3 describes current methods of improvement location and preference. Chapter 4 describes how they are currently intertwined.

Current local search methods do not independently compare methods of improvement location and preference. Current research does not systematically distinguish between how improvements are located and which improvements are given preference.

Hypothesis

Distinguishing between improvement location and improvement preference is feasible and useful when creating local search algorithms.

The hypothesis is discussed in depth in Chapter 4.

2) Background and Key Concepts

The vast number of possible solutions to combinatorial optimisation problems means we sometimes have to use heuristics. A heuristic is in effect an algorithm that makes a series of educated guesses. Heuristics use educated guesses to enable them to find good quality solutions within the time available.

Heuristics vary in speed. At one extreme we have greedy algorithms, they produce solutions very quickly, but the solution quality tends to have plenty of room left for improvement. Greedy algorithms construct solutions from scratch. They construct solutions by repeatedly adding what looks to be the best feasible problem element, until the solution is complete. At the other extreme are local search heuristics that take a solution and make a series of improvements to it. These typically need more processing time and tend to produce higher quality solutions.

Different local search methods use different methods of locating improvements. How these improvements are located affects execution time and solution quality. While the exact details of how they locate improvements differ from method to method, they all use solution changes to locate improvements. All local search methods take an existing solution and change it to create a new solution. The quality of the new solution and the old solution are then compared. The difference in quality is used to decide if the new solution should be used to try and locate further improvements. The benefits of using different methods of locating improvements are tested in the experiments.

Local search methods use different methods of guiding the improvement preference. Some methods give preference to large improvements while others give preference to non-improving solutions. The order which improvements are implemented affects execution time and solution quality. Some local search methods use both improving and non-improving solutions to guide the improvement preference. They use the quality of the improving or non-improving change to decide if the change should be use to locate improvements. Two of the best known local search methods are Simulated Annealing (SA) and Tabu Search (TS), which use two different methods of guiding improvement preference, for a description of SA see [Dowsland 93] and for TS see [Glover 97]. The benefits of using different methods of guiding the improvement preference are tested in my experiments.

This chapter gives a overview of combinatorial problems and the heuristics used to tackle them.

2.1) The TSP

The Travelling Salesman Problem (TSP) is used as an example problem to explain the key concepts.

Figure 2.1 - TSP solution – one possible route the sales person could take

The Travelling Salesman Problem is a problem where a sales-person needs to visit a number of customers. The dots in Figure 2.1 indicate customers that the sales-person needs to visit and each line indicates a journey between two customers. The aim is to try and work out the shortest route the salesperson can take in order to visit each customer once and return to the original start point. The Travelling Salesman Problem (TSP) is a combinatorial optimisation problem and a large number of heuristics have been created for it. See [Johnson 97] for a review of TSP heuristics.

With large numbers of customers it becomes impossible to search through the huge number of possible solutions; i.e. ((n-1)!)/2, where n is the number of customers. If the problem is small enough then methods such as branch and bound can be used, see [Madsen 98] for an description of branch and bound. Branch and bound does not search through all ((n-1)!)/2 solutions to find the shortest as it is able to rule out many possible solutions. Although methods such as branch and bound are unable to guarantee they will find the shortest route in polynomial execution time. There is not always enough time to wait for an algorithm to find the shortest route. With large problems a local search heuristic is often preferred and is used to find a route quite close in length to the shortest.

2.2) Using a hill climber to solve the TSP

A hill climber is a simple local search heuristic. It takes a solution to a combinatorial problem and makes a series of improvements to it. The method makes no guarantees about how good the end solution will be, as it is not designed to check for all possible improvements.

When a change is made to a solution, this is called a move, i.e. we move from one solution to another. Figure 2.2 shows two solutions, the top one has 2 lines that cross, by removing these two lines and then completing the TSP solution again by inserting two new lines a new solution is created. The two inserted lines are shown in the new improved solution at the bottom of figure 2.2. The solution is an improvement because with the lower solution the sales-person has a shorter journey. The path between two customers is known as an edge. This act of removing two edges (lines) and inserting a different two is known as a 2-opt move. With the TSP, the simplest, least disruptive move that can be made is a 2-opt move. For a history of 2-opt see [Applegate 99]. A 2-opt is when a pair of edges (lines) between customers is removed and replaced by the alternative pair of edges.

The 2 in 2-opt refers to the number of edges (paths) that are changed when making improvements to an initial solution. By repeatedly making 2-opt improvements to an initial solution, a sub-optimal solution is created, this solution is classed as 2-optimal, hence the term 2-opt. A 3-opt means 3 edges are removed and 3 are inserted to create a new solution, and an n-opt means n edges are removed and inserted.

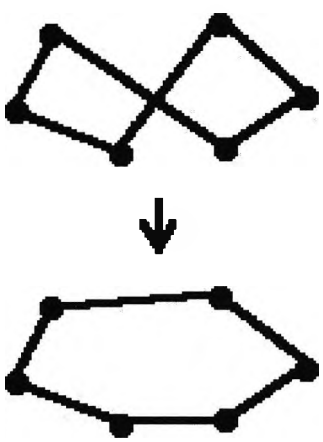


Figure 2.2 – 2-opt move operator, changes two edges, producing a new solution

A 2-opt hill climber will often fail to find the shortest route, but is normally able to find a solution quite close to the shortest, see [Johnson 97].

```
/* Hill Climber Example */

create an initial random solution
note initial solution is best solution so far
WHILE untested small solution changes exist
    make a small untried change to the current best solution
    IF the small change produces an improvement
        make note of new best solution
    ENDIF
ENDWHILE
```

Figure 2.3 - Hill Climber Pseudo Code

The hill climber, see Figure 2.3 for pseudo code, works as follows: First a solution to the problem is created, any random solution will do. Next a small part of the solution is changed to create a slightly different solution, if the changed solution is better than the original then it is used to create new solutions. If it is not better it is rejected and a different change is made to the original solution. Because only small changes (moves) are made to the current best solution, at some point none of the available moves will produce an improvement. When none of the small changes produce an improvement the solution is known as a local optimum. The search process is normally stopped when a local optimum is reached.

All the available small changes that can be made to a solution are known as the neighbourhood of the solution. Local search typically uses small changes as this keeps the number of solutions in the neighbourhood low.

A hill climber is a kind of search control, i.e. it controls the search for new solutions. It decides which new solutions are used and which are discarded.

When a hill climber is used on a TSP, unless the problem is very small, most of the time it will not find the shortest route. Optimal solutions, i.e. the shortest routes, are known as the global optima. A hill climber reaches what is known as a local optima when it is unable to find further improvements and yet does not find the global optima. The point of using a hill climber on a TSP is not to find the shortest route, as this would take too long. A hill climber uses the time available to try and find a route that is quite close to the shortest.

Because the optimal solution to a TSP forms a polygon and the paths between customers form the edges of the polygon, the path between two customers is often referred to as an edge.

2.2.1) Vehicle Routing Problem with Time Windows and Capacity Constraints

The basic VRP is when goods need delivering to many different customers using a fleet of vehicles, see Figure 5.4. This problem is in some respects similar to the Travelling Salesman Problem (TSP), but it also has constraints that the TSP does not have.

- As with the TSP, adjacent customers are used to calculate cost and so we must control disruptions to adjacency.

- Capacity constraints put a limit on the maximum load we can put in each vehicle, normally a maximum volume or weight. Capacity constraints constrain which customer deliveries can be grouped together. The problem style becomes a combination of adjacency and group.
- Time windows occur when each customer has a time slot stating when the goods can be delivered. This adds yet another dimension to evaluation as the order customers are visited affects the ability to hit time windows, so we must control disruptions to precedence. The problem style becomes a combination of adjacency, grouping and precedence.

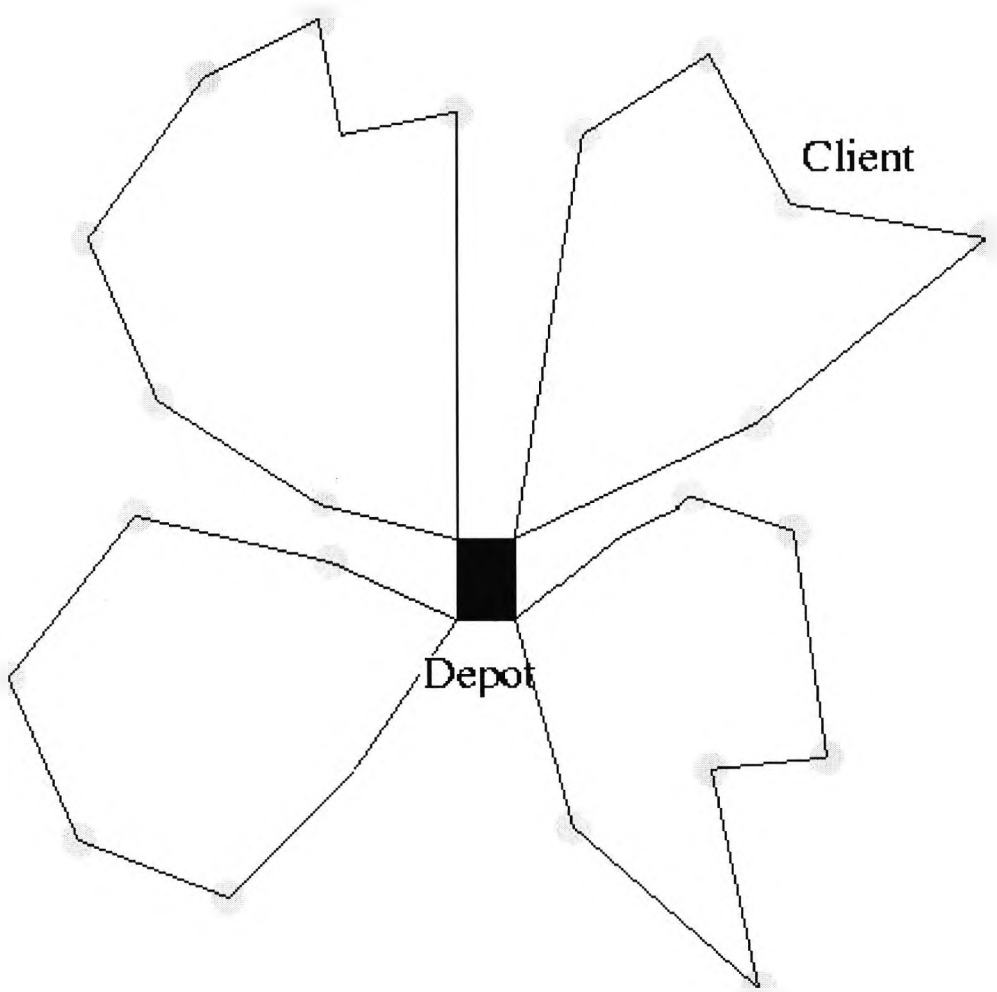


Figure 2.4 -Vehicle Routing Problem

Other common VRP constraints include more than one depot, multiple time windows per customer, loads larger than a single vehicle, pickups and driver breaks. For a real case VRP that uses tabu search see [Rochat 94], which describes in detail many real constrains and how they were dealt with.

2.3) NP-hardness

Roughly speaking this means the time needed to find the shortest route becomes excessively large with larger problem sizes. The existing methods that guarantee to find the optimal solution to all TSP problems have a non-polynomial running time. Because it has not been proved if the TSP *can* or *can not* always be solved in polynomial time, it is in class NP, meaning Non-deterministic Polynomial. When an algorithm can not execute in polynormial time the term Non-polynomial time is used, it means the execution time needed by the algorithm is not a polynomial function of the problem size. There are many optimisation problems that are NP-hard, including Vehicle Routing Problems, Bin packing problems and Scheduling problems. More information on NP-hardness and intractability can be found in [Garey 1979].

The algorithms being considered here are designed to create high-quality non-optimal solutions. These algorithms are used when there is not enough time available to find an optimal solution. There are exact methods capable of finding optimal solutions to problems that are NP-hard [Madsen 98]. Exact methods execute in a reasonable length of time if the problem is small enough. Although with many problems that are NP-hard it is not important to find the optimal solution, a less than optimal solution that is found faster is often preferred. The thesis is only considering local search methods where the trade off between execution time and solution quality matters.

2.4) Heuristics

Heuristic are experimental rules that are not necessarily provably correct but typically produce acceptable solutions. Heuristics usually do not guarantee how close to the global optimum the final solution will be. Instead they use a series of educated guesses to find high quality solutions. For example a hill climber is a heuristic that is sometimes able to find acceptable solutions to NP-hard optimisation problems, see [Reeves 93].

The Artificial Intelligence (AI) community and the Operational Research (OR) community use slightly different definitions of the word heuristic. In AI an algorithm is said to use a heuristic to guide the search [Russell 95], whereas in OR a heuristic is an algorithm that is not guaranteed to find the optimal solution [Reeves 93] (in AI this is termed incomplete search).

The term Meta-heuristic means a general heuristic that can be applied to more than one problem. A meta-heuristic is a kind of high level framework that can be use to create a heuristic for a specific problem. There are two main styles of meta-heuristic for combinatorial problems, construction and local search.

2.4.1) Construction Heuristics

Construction heuristics create solutions from scratch and can be very fast. A simple construction method for the TSP is to pick a customer at random then keep travelling to the nearest unvisited customer until all the customers have been visited. This method is often called a greedy method. This is because it has a short-term 'greedy' strategy of picking the best element available when inserting elements into

the solution. Figure 2.5 gives example pseudo code for a greedy construction heuristic.

```
/* Construction heuristic example */

start off without any solution properties set, a property is a relationship
        between two or more elements of the problem, such as an edge.

DO WHILE solution is incomplete
    compare costs of some or all feasible solution properties
    add property with least cost
ENDDO
```

Figure 2.5 - Greedy Construction Heuristic Pseudo Code

More complex construction algorithms exist for the TSP and these are able to create better quality solutions. A comparison of TSP construction algorithms can be seen in [Johnson 97] and [Johnson 02]. The best construction methods, in terms of solution quality, for the TSP are variations on Christofides algorithm [Johnson 02], they can typically get to within 7-9% of the Held-Karp lower bound, see [Held 70], [Held 71] and [Johnson 96]. The Held-Karp lower bound is normally slightly shorter than the optimal TSP solution, i.e. feasible solutions can not be shorter than the lower bound.

2.4.2) Local search

A method of changing the current solution and a methods of deciding which changes to accept or reject make up the two basic elements of local search.

Local search heuristics normally change only a small, localised part of the solution, leaving most of it untouched. They often use a construction heuristic to create one or

more initial solutions and then repeatedly attempt to make improvements to the solution(s).

The aim of local search is to take an existing solution and improve it. Because of this local search heuristics are some times known as Iterative Improvement heuristics [Zweben 90], [Zweben 94]. They take an existing solution and make a series of improvements to it.

Hill climbers in contrast with other local search heuristics do not use non-improving solutions to find improvements. The other local search methods that use non-improving solutions to find improvements are more complex. Well known local search methods that use non-improving solutions include Tabu Search, Simulated Annealing and Evolutionary Algorithms. A description of different ways of implementing these can be found in [Reeves 93].

2.5) Summary

Local search heuristics construct a solution then make a series of improvements to it. The focus of my experiments is to compare methods of locating improvements and guiding the improvement preference. This thesis examines NP-hard problems where it is important to find the best quality solution possible in the time available rather than the optimal solution.

3) Review of Local Search Methods

Because current methods of local search do not distinguish between the problems of improvement location and preference it is argued in this chapter and Chapter 4 that there is a gap in our current knowledge. The review describes how existing local search meta-heuristics locate improvements and how they give preference to some improvements rather than others. The methods described do not distinguish between location and preference making it unclear how much impact each is having.

Emphasis is placed on how similar or different the methods are. Untraditional descriptions are sometimes used to emphasise similarity. The review roughly categorises the major methods of location and preference. These categories are then used to argue that Lin-Kernighan [Lin 73], a very successful method of improvement location, is under used. Distinguishing between location and preference helped identify this underused, yet very successful method. This is used to argue that distinguishing between location and preference is useful.

Local search methods use non-improving solutions to find improvements. The descriptions have been skewed in order to emphasise how the methods use non-improving moves. Lin-Kernighan is highlighted because unlike other local search methods it uses partial solutions to find improvements.

This chapter aims to describe the current major methods of guiding the local search process and covers: -

- Several methods that use whole solutions to guide the search for improvements and one method that does not.
- Local search methods that show different ways of guiding the search.
- Some recent developments and new ideas that offer ways to improve algorithm speed and solution quality.
- Move operator design, which relates to the move operators used in the experiments.

There are many heuristics for tackling combinatorial optimisation problems. One of the oldest and most successful is Lin-Kernighan [Lin 73], the basic algorithm still dominates the Travelling Salesman Problem (TSP) as can be seen in [Johnson 02]. Another successful method, Iterated Local Search, is increasingly being used to improve the final solution quality of existing local search methods when extra time is available, see [Lorenco 02].

There are many heuristics that have established reputations for being able to find good quality solutions to combinatorial problems. The major ones are described later in this chapter; for more in depth descriptions see [Reeves 93], [Johnson 97], [Dorigo 02]. Ant Colony Optimisation is a more recent heuristic, [Dorigo 91], and it appears to be receiving quite a bit of attention, for an overview see [Dorigo 99] and for some recent developments see [Dorigo 02]. For the TSP there are hybrids between population based methods and Iterated Lin-Kernighan that offer further

small improvements in quality but this comes at the expense of large increases in execution time, see [Applegate 99], [Johnson 97].

3.1) Broad overview of major methods

This section aims to show that a large variety of different local search methods exist. On the surface these methods appear quite different from each other, their competing differences and similarities are highlighted in this section. The experiments chapter describes how competing ideas of how to guide the search process are compared.

3.1.1) Hill Climbers, stochastic and steepest ascent

A hill climber is the most basic form of search control, it only allows improvements to be made to a solution. Different forms of hill climber exist and they use different rules for working out which improvements to make next. Probably the two most common are: -

- Stochastic Hill Climber - randomly evaluates moves in the neighbourhood, if the move is an improvement, it implements it. See Figure 3.1 below.
- Steepest Ascent Hill Climber – is a computationally expensive hill climber as it involves the evaluation of every move in the neighbourhood and then implements the most improving. See Figure 3.2 below.

```
/* Stochastic Hill Climber Example */

create an initial random solution
note initial solution is best solution so far
WHILE untested small solution changes exist

    make a small untried change to the current best solution, randomly
            selected from the untried changes in the neighbourhood.

    IF the small change produces an improvement
        make note of new best solution
    ENDIF

ENDWHILE
```

Figure 3.1 - Stochastic Hill Climber Pseudo Code

```
/* Steepest Ascent Hill Climber Example */

create an initial random solution
note initial solution is best solution so far
DO
    create a list of the entire set of solutions that can be created by making
            small changes to the current best solution

    make a note of the current best solution so we can tell if an improvement
            has been found
    DO FOR all small solution changes in list
        IF solution is better than best current solution
            make solution the best current solution
        ENDIF
    ENDDO

WHILE improvement found
```

Figure 3.2 - Steepest Ascent Hill Climber Pseudo Code

Stochastic hill climbers are sometimes used as a benchmark [Juels 94]. Other search control methods are compared with hill climbers to assess performance. A stochastic

hill climber uses random move selection to find the next improvement. This makes it a handy method for comparing more complex methods against.

In experiments comparing a stochastic hill climber with steepest ascent, stochastic hill climbing was not only faster but also achieved better quality solutions, see [Tuson 2000]. While this is just a single set of experiments is does show that steepest ascent is not always the best choice.

Relationship between Hill Climbers and Hypothesis. See section 4.1 for hypothesis, and see Overview Table 3.1

Stochastic Hill Climber

- Uses the differences between whole solutions to guide the search for improvements
- Improvements are located using a fixed size move operator
- No preference is given to improving a particular part of the current solution, resulting in no improvement preference.

Steepest ascent Hill Climber

- Uses the differences between whole solutions to guide the search for improvements
- Improvements are located using a fixed size move operator
- The most improving part of the neighbourhood is always improved at each step resulting in improvements being implemented using a largest improvement first preference.

3.1.2) Tabu Search

Tabu Search was devised by Fred Glover, see [Glover 97], and is described as having four types of memory, Recency, Frequency, Quality and Influence. Recency and Frequency memory are lists of what has been tried and are used to guide the search to new areas of the search space. The meaning and use of quality memory and influence memory is less well defined. [Glover 97] describes Quality memory and Influence memory as being about more than cost (i.e. solution quality), although there does not appear to be a detailed description of what is meant by this. The examples given in [Glover 97] only make use of cost differences.

Tabu Search uses a steepest ascent hill climber until it reaches a local optima, see Figure 3.3 for pseudo code. It then implements the move with the smallest non-improving quality in the neighbourhood. It then makes some element of this move Tabu, so as to stop it being undone, this is stored in the Recency list. It then looks for and implements any improvements it can find. When there are no more non-Tabu improvements left in the neighbourhood it will repeat the process, implementing the smallest non-improving move, make it Tabu, and so on.

A limit is imposed on how long move elements remain Tabu, this allows moves to be undone and redone. To prevent the search looping Tabu Search uses a Frequency list to count how many times a move has been made Tabu and prevents the non-improving move from being made again if the count is too high. Frequency prevents improving moves from being undone and controls search width. Recency prevents recent non-improving moves from being undone and controls search depth.

Many ideas about Tabu Search and local search heuristics can be found in [Glover 97]. One such idea is candidate lists. When Tabu Search uses steepest ascent it can be quite slow because it needs to evaluate the entire neighbourhood, candidate lists are one way of getting round this problem. There is a description of candidate lists and other ways to improve the speed of Tabu Search later in this chapter.

```
/* Tabu Search Example */

create an initial random solution
note initial solution is current solution
note initial solution is best solution found so far
create an empty list for keeping a record of recent solution changes
create an empty list for keeping a record of the frequency of changes

DO
    create a list of the entire set of solutions that can be created by making
            small changes to the current best solution

    make a note of the current best solution so we can tell if an improvement
            has been found

    DO FOR all small solution changes in list
        IF solution is better than best solution found so far
            make note solution is the new best solution found so far
            make note solution is also best found so far in neighbourhood
        ELSE
            IF solution is better than best found so far in neighbourhood
                IF solution is better than current solution
                    IF move was not part of a recent back move
                        note solution is best found so far in neighbourhood
                    ENDIF
                ELSE
                    IF move has not been made and undone too frequently
                        note solution is best found so far in neighbourhood
                    ENDIF
                ENDIF
            ENDIF
        ENDIF
        IF best neighbour is better than current solution
            add best neighbour to frequency list
        ELSE
            add best neighbour to recency list
        ENDIF
        set current solution to best neighbour in neighbourhood
    ENDDO

WHILE typically stop after some fixed number of loops round
```

Figure 3.3 – Tabu Search Pseudo Code

Relationship between Tabu Search and Hypothesis. See Table 3.1

- Uses the differences between whole solutions to guide the search for improvements
- Improvements are located using a fixed size move operator and individual whole non-improving solutions.
- The most improving part of the neighbourhood is always improved at each step resulting in improvements being implemented using a largest first preference. Improvements located using non-improving solutions that are of much worse quality than the current best solution are implemented at the end of the optimisation process. Improvements located using a large number of compounded non-improving moves are looked for and implemented at the end of the optimisation process.

3.1.3) Simulated Annealing and Threshold methods

Simulated Annealing, [Kirkpatrick 83], is a stochastic hill climber that from time to time also makes non-improving moves inspired by processes in statistical mechanics in physics. See [Dowsland 93] for a more detailed description of Simulated Annealing. All improving moves found are implemented and a biased random selection method is used to work out which non-improving moves to implement. Selection is biased towards changes that make the solution a little worse. As the search progresses it increases its bias against moves that are worse than the current best solution, until it reaches a stage when only improving changes are accepted.

Annealing refers to the cooling of some material. The rate at which the bias is increase against non-improving solutions relates to the speed the temperature is decreased.

Simulated Annealing Algorithm – for minimisation problem

Where t is the temperature, the initial temperature is set to some value >0, and gradually reduced to 0.

```
d = new solution quality - current solution quality

if exp(-d/t) > random number between 1..0 then
        current solution = new solution
```

Note if the new solution is an improvement, i.e. d is negative, then it is always accepted.

The rate at which the temperature is reduced has traditionally been worked out using trial and error. A method that automatically calculates the temperature reduction rate and thus the convergence rate for SA and Evolutionary Algorithms, [Poupaert 2001], is described later in this chapter.

Threshold methods

Threshold methods are fairly similar to simulated annealing, but rather than increasing the bias against non-improving changes they gradually increase an acceptance threshold. This means they reject more and more non-improving changes as the search progresses. Various styles of threshold have been experimented with, methods where the threshold is: -

- Relative to the current solution – Threshold Accepting [Dueck 90]
- Relative to the current best solution – Record To Record Travel [Dueck 93]
- An absolute solution quality threshold – Great Deluge Algorithm [Dueck 93]

Relationship between Simulated Annealing and Hypothesis. See Table 3.1

- Uses the differences between whole solutions to guide the search for improvements
- Improvements are located using a fixed size move operator and individual whole non-improving solutions
- All improvements found are implemented, resulting in a bias towards no improvements implemented preference. Improvements located using non-improving solutions that are much worse quality than the current best solution are implemented at the start of the search process. Improvements located using a large number of compounded non-improving moves are looked for and implemented at the start of the search process.

3.1.4) Evolutionary Algorithms

The term Evolutionary Algorithm (EA) is used in the text as a general term to describe both Genetic Algorithms and other evolution based algorithms. EAs have been around for more than 40 years, [Bremermann 62]. EAs use a population of solutions and are broadly based on biological evolution.

Evolutionary Algorithms can be split into three major groups: -

- Genetic Algorithms see [Goldberg 89], [Holland 75]
- Evolutionary Programming see [Fogel 95], [Fogel 66]
- Evolutionary Strategies see [Back 96], [Rechenberg 73]

The following text describes the basic framework of local search style EAs.

Population
This is a pool of solutions used to create new solutions. The initial population of solutions can be created in a number of ways. Random solutions can be created, alternatively meta-heuristics such as a greedy construction method or a local search method can be used to create the initial population. It is important that population members are different from each other so the method can create better solutions. Population size affects algorithm speed and solution quality, see the genetic algorithms chapter in [Reeves 93] for a brief discussion on population size.

Selection
The selection process chooses which solutions to discard and which solutions to use for reproduction, where reproduction is taking one or more solutions and using the solutions to create a new solution. The simplest way to choose a solution for reproduction is to use solution quality, often called fitness. Weaker, low quality solutions are discarded and better quality solutions are used to create new solutions.

One method of selection is to replace the current generation of solutions with a new generation, this is known as generational replacement [Goldberg 89]. This method sometimes leads to the current best solution being discarded before it has out lived its usefulness. Another method is to gradually replace weaker members of the population with new stronger solutions, this is known as steady-state replacement, see [Bäck 97].

Selection can also be based on how similar solutions are, i.e. how many solution properties are the same. When solutions are too similar to other members of the population they are discarded. Selection can control diversity by using solutions that are or are not similar to create new solutions.

Crossover

Crossover is a method of reproduction, for example see [Michalewicz 02].

Crossover is when properties are copied from two solutions to create a new solution. For example, with the Travelling Salesman Problem (TSP) take two solutions that have slightly different routes. Some of the paths between customers will differ, allowing us to create a new solution using some paths from one solution and some from the other. Crossover normally involves using two existing solutions but can involve more. Properties that are the same in both solutions are normally left unchanged where possible. It is the properties that are different that are changed and exchanged in order to create a new solution. The aim is to create better solutions by combining properties from existing good quality solutions.

Mutation

Mutation is a method of reproduction, for example see [Michalewicz 02]. Mutation is when the Evolutionary Algorithm takes a single solution and randomly changes some of its properties to create a new solution. The mutation does not have to be random but there is usually some random factor involved. Mutation is used to help find solutions that are better than the current best solution. These mutations will sometimes manage to indirectly find improvements but sometimes they will fail to help improve the current best solution.

```
/* Evolutionary Algorithm Example */

create an initial random solution
note initial solution is best solution found so far

create a list to hold current population of solution

put initial solution in the population list

DO

    search population for solutions that are worth using cross over on. Note
select these based on number of differences between solutions (move
operator size) and difference in solution quality.

    IF solutions for cross over found
        create a new solution using cross over
    ELSE
        select a solution from the population to mutate
        use a move (mutation) operator on the solution creating new solution
    ENDIF

    add the new solution to population list

    IF the new solution is better then the current best solution
        make note new solution is the current best solution
    ENDIF

    work through current population list discarding any unpromising solutions

WHILE stop after a fixed number of loops or several failures to find
improvements
```

Figure 3.4 - Evolutionary Algorithm Pseudo Code

Putting it all together
A collection of solutions is created, this collection is called a population. Two solutions are selected and some of the properties that exist in one of the solutions get copied into the other to create a new solution. Good quality solutions have some good quality properties; the idea is to combine compatible good quality properties into a single better quality solution.

Merging good quality solution properties into one solution has the problem of convergence. This is when all the best solutions become very similar to each other. Mutation can be used to control convergence by controlling the introduction of new solution properties.

Tuning of convergence has traditionally been done by trial and error. This is done by changing the population size, mutation rate and the way solutions are discarded etc. A method of controlling convergence for EAs and SA, [Poupaert 2001], is described later in this chapter.

Evolutionary Algorithm Example
The example given in Figure 3.4 starts off with a single solution, rather than a population of very different solutions. The pseudo code is able to produce a population of very different or very similar solutions by controlling the amount of solution mutation. Thus a population of very different solutions is a design choice and a population of very similar solutions could be created. This can be seen as being like the neighbourhoods of Simulated Annealing and Tabu Search.

Relationship between Evolutionary Algorithm and Hypothesis. See Table 3.1

- Uses the differences between whole solutions to guide the search for improvements
- Improvements are located using variable size move operators and many whole non-improving solutions
- There is typically a bias toward using non-improving solutions that are only a little worse than the current best solution to locate improvements. Such improvements tend to be implemented at the start of the search process.

3.1.5) Ant Colonies

Ant Colonies proposed by [Dorigo 91], like EAs, use more than one solution to create new solutions. Ant Colonies use a model to guide the creation of new solutions and that model is created using many whole solutions. The term Ant Colonies (AC) is used in the text as a general term, although different variations of the method have often been given slightly different names.

AC combine several good quality solutions to produce new solutions. The method takes several solutions and works out which bits of the solutions are the same. When many different solutions share a particular property, its value is calculated using the qualities of all the solutions it is part of. So if five solutions have a solution property in common then the property is awarded a value based on the qualities of the five solutions.

For example, with the TSP, if five solutions all use the same edge, that edge is given a value (weighting) based on the qualities of the five solutions. Solution properties are given weightings based on the solutions that share the property. When creating

new solution, solution properties (edges) are selected on a probabilistic basis, according to weighting. As a result the knowledge gained from old solutions does not need to be thrown away, as it can be stored in the weighting. This means a population of solutions does not need to be maintained, as only the weightings are needed.

To control convergence, weightings from older solutions are reduced each time new solutions are created. This encourages new solutions that are slightly worse than the existing solution to be used. This means the new solutions get used to try and find improvements.

With EAs there is the problem of working out which solutions to combine together. The advantage with AC is the appropriate solution properties are available at each choice point, along with estimates of their value. This removes the problem of working out which solutions to use to create a new solution. See [Dorigo 99] for a more complete description of AC.

AC, unlike the other methods described here, store the estimate of a property's value for future use. AC, like the other local search meta-heuristics, use the difference in quality between whole solutions to estimate the value of the properties that are different. These estimates are then used to guide the search for improvements.

Relationship between Ant Colonies and Hypothesis. See Table 3.1

- Uses the differences between whole solutions to guide the search for improvements
- Improvements are located using variable size move operators and many whole non-improving solutions.
- There is a bias toward using non-improving solutions that are only a little worse than the current best solution to locate improvements. Such improvements tend to be implemented at the start of the search process.

3.1.6) Iterated Local Search

Iterated Local Search has proved a valuable way of further improving solutions created by local search heuristics, [Lorenco 02], [Johnson 02], [Martin 91]. It takes a solution produced by a local search heuristic, then uses a 'kick' that makes the solution worse, then re-runs the local search heuristic. For example [Martin 91] 'kicks' TSP solutions using a double bridge, essentially this involves splitting the route into two routes and then joining them together again elsewhere. The kick is defined by [Martin 91] as, the smallest move that is not easy to create using the existing move operator. The kick is repeatedly applied to the best solution found so far, followed by the local search heuristic. While other methods have been tried, always applying the kick to the current best solution appears to be the most successful [Lorenco 02]. A comparison of different styles of implementing iterated local search is made in [Lorenco 02].

3.2) Enhancements to local search methods

This section describes how some of the most widely used local search methods have been extended and improved.

3.2.1) Tabu Search Speed Ups

Tabu Search typically evaluates all the solutions in the neighbourhood to find the best improvement, this is computationally expensive. Instead of re-evaluating the entire neighbourhood after a move has been implemented speed can be improved by re-evaluating only the neighbours that are impacted by the change. An examination of some ways to speed up steepest ascent within Tabu Search can be found in [Johnson 97]. Even with these speedups Tabu Search appears to be a long way behind the other TSP heuristics that Johnson looked at. One of the faster Tabu Search methods looked at by Johnson was about 800 times slower than a 3-opt hill climber and still ended up with a worse average solution.

Tabu Search interestingly appears to be the local search method of choice for the Vehicle Routing Problem, [Cordeau 02], although the paper does not give any reasons as to why it is so popular. Granular Tabu Search, [Toth 98], speeds up the search process by removing unpromising edges from the problem before it starts the search process. The method leaves 10-20% of the original edges in the problem. It appears to be 3-5 times faster than other Tabu Search methods that produce similar quality solutions, see [Cordeau 02] for a comparison of Vehicle Routing Problem methods.

Glover [Glover 97] suggests the use of candidate lists, where a subset of the neighbourhood is evaluated and the best of move in the subset is implemented. Any ascent was able to outperform candidate lists and best ascent in [Anderson 96]. Experimental results, in [Anderson 96], on the TSP suggest candidate lists and best ascent are less effective than any ascent. This is because repeated use any ascent was able to find better solutions in less time than repeated use of candidate lists or best ascent.

3.2.2) Evolutionary Algorithm Hybrids

Many Evolutionary Algorithm (EA) hybrids exist, such hybrids have resulted in improvements in speed and/or quality see [Chellapilla 98] and [Johnson 97]. In one paper an EA hybrid failed to outperform an EA. The case was in [Braun 99] and combines an EA with Simulated Annealing. Although the implementation of the hybrid is suspect, as the algorithm typically made the initial greedy solution worse rather than better. Typically EA hybrids appear to improve EA performance.

One of the simplest forms of hybrid is to use a greedy method to create the initial population. Several different styles of initial solution were used by [Braun 99]. The best quality results were achieved by using the best of the initial solutions.

A hybrid between SA and EA uses SA to control acceptance of EA mutations, sometimes called SAGA (simulated annealing genetic algorithm). This can be used to control the convergence rate of the EA, see [Poupaert 01], [Tang 00] and [Braun 99]. [Poupaert 01] is described in more depth in the next section.

Another EA hybrid involves using a local search method to improve members of the population after crossover. These range from using simple hill climbers such as in [Tang 00], to using Iterated Lin-Kernighan (ILK) to improve members of the population. [Johnson 97] reports this is faster at finding improvements than could be achieved with repeated use of ILK.

A method that combined Tabu Search, Simulated Annealing and Genetic Algorithms was created in [Ozdamar 02]. Although it is hard to tell if this offered any benefits as each of the methods compared had different running times. The methods used Tabu lists, isolating SA acceptance criteria and multiple population GAs.

Stochastic hill climbers have been used to improve the design of Evolutionary Algorithms, see [Juels 94]. A stochastic hill climber was created to tackle the problem and the knowledge gained was used to improve the design of the Evolutionary Algorithm.

Relationship between SAGA and Hypothesis. See Table 3.1

- Uses the differences between whole solutions to guide the search for improvements
- Improvements are located using variable size move operators and many whole non-improving solutions.
- Non-improving solutions that are much worse than the current best solution are used to locate improvements at the start of the search.

3.2.3) Controlling convergence in SA & EA

Using a convergence rule from SA [Poupaert 2001] demonstrated a successful way of controlling execution time using self-tuning convergence. The convergence rule from SA says, "the number of accepted SA moves must be constant for each temperature", see [Poupaert 2001]. This results in the search having to search through more and more non-improving solutions as it progresses.

As with traditional SA, the worse a solution's quality, the higher the chance it will be discarded, this bias against worse solutions is increased as the search progresses. Changing the amount of bias against worse solutions, changes execution time. With Acceptance Driven Selection [Poupaert 2001] you get to enter the desired execution time. The method will then use the results of past evaluations to work out which solutions to discard. This allows it to reach convergence at a time close to the desired execution time. The above convergence rule and past evaluations were used to create a formula to estimate the acceptance level as the search progressed, see [Poupaert 2001].

Poupaert compares several different implementations of EA and SA based on this method of convergence. The method reports to perform better than standard SA and EA as well as matching the best SA algorithms. The big advantage of the method is that you state how long you want it to run for and it will adjust its convergence rate accordingly.

3.2.4) Ant Colony Enhancements

One enhancement to ant colonies is to give extra weighting to properties that exist in the best solution, giving properties of the best solution a higher chance of being used again, see [Dorigo 02]. Another method along similar lines is to create several solutions, rank them based on quality and then only use the best of these solutions to create weightings, biasing the weighing according rank.

One problem is that ants will often select the same properties as each other and end up building very similar solutions. To get round this a method called Ant Colony System [Dorigo 97] reduces the weight for a property when it gets used. This means that other ants will then have a higher chance of using other promising solution properties not yet used.

3.3) Move operator design – Absolute order, relative order & position

This section looks at definitions of different types of solution property that are relevant to sequencing, routing and assignment problems.

Several problem categories have been defined [Bierwirth 1996], [Tuson 2000]. The relationships between solution elements have been used to categorise different styles of problem. The quality and feasibility of a solution is calculated based on the relationships that exist between solution elements. These relationships are the solution properties that are used to estimate the value of a change to a solution.

Different problem styles have different types of solution properties. Three well-known problem styles are precedence, adjacency & group, see bullet points below. For an in-depth look at move operator performance on scheduling problems see [Bierwirth 1996], a discussion of move operators and permutation encoding can be found in [Tuson 2000]. Absolute order, relative order & position are described in [Bierwirth 1996] and [Tuson 2000]. Asymmetrical adjacency is also described, although it is similar to precedence.

The move operators described can be used to change a single solution or used when copying properties from one solution to another.

- Precedence/Absolute order, shift operator – e.g. scheduling

This is deciding what order to do some set of tasks, such as scheduling production on a production line. The tasks that come before a particular task determine that task's possible cost. What matters is the order tasks are completed. So with any two tasks in the schedule, the relationship that matters is which one comes first. It is this relationship that local search methods improve. Common constraints include, tasks being prerequisites of other tasks, desired task completion times. For an example of experiments that improve scheduling problem quality and various scheduling speed up methods see [Congram 1998].

Shift operators tend to be used with this kind of problem as they can control the disruption of which task comes first. A shift operator takes a task in the sequence and repositions it at a new location. Experiments comparing this kind of operator

against others can be found in [Tuson 2000]. [Bierwirth 1996] looks at crossover operators for a scheduling problem.

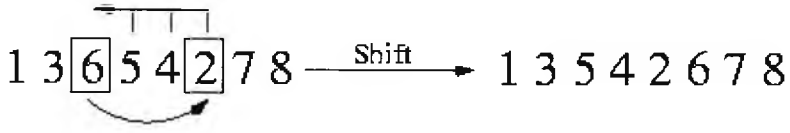


Figure 3.5 – Shift Operator – Reprinted with permission from [Tuson 00]

- Adjacency/Relative order, 2-opt – e.g. TSP

Adjacency is common in routing problems, problems such as TSP. It is about deciding which tasks should be adjacent. Reversing the order of the tasks has no impact on cost, unlike precedence. It is the pairs of adjacent tasks that determine cost, so it does not matter which tasks come first.

Scheduling problems can also have this property. A scheduling problem where the only variable cost is switching from one task to another and the direction of the switch does not affect cost is also of this type.

A k-opt move operator tends to be used with this style of problem, as it can control disruption of adjacent tasks. 2-opt reverses a sub section of the route or list, a k-opt reverses k-1 sub sections. For a description of the history of 2-opt and 3-opt, developed in the 1950's and 60's, see [Applegate 99].

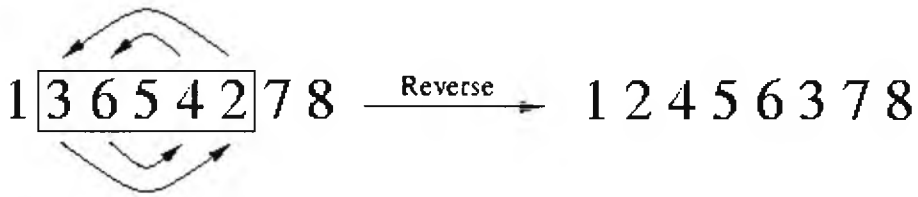


Figure 3.6 – 2-Opt Operator – Reverse – Reprinted with permission from [Tuson 00]

- Group/Position, swap operator – e.g. assignment

This involves grouping elements or tasks together, such as bin packing or designing a school timetable. It is which elements end up in the same group that determines cost. Common constraints include, restrictions placed on group sizes, extra costs when some tasks are in the same group, tasks having different costs depending on what group they are in.

A swap operator tends to be used with this style of problem, as it controls disruption of elements that are in the same group. A swap operator switches elements between groups.

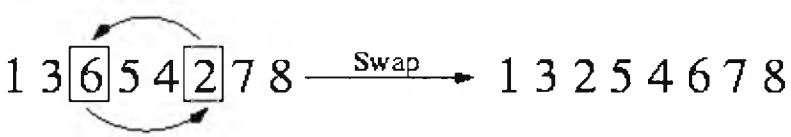


Figure 3.7 – Swap Operator – Reprinted with permission from [Tuson 00]

- Asymmetrical adjacency, shift operator – ATSP

This concerns the ordering of adjacent tasks. Tasks such as the Asymmetrical Travelling Salesman Problem (ATSP) where the distance between A and B is different from the distance between B and A. This could, for example, be caused by

a one way traffic system. Scheduling problems can have this property where the cost of switching from task A to task B is different cost to the cost of switching from B to A. This can be seen as being half way between precedence and adjacency, as the precedence of the adjacent elements matters. It is the order of adjacent elements determines cost. As with precedence a shift operator is useful for this style of problem as it can control disruption to the order of adjacent elements.

3.4) Lin-Kernighan Algorithms

The dominance of Lin-Kernighan and Lin-Kernighan hybrids with the TSP can be seen in [Johnson 97] and [Johnson 02]. When it comes to finding solutions to the TSP, Lin-Kernighan and Lin-Kernighan hybrids appear to be faster than any of the other local search algorithms. Construction methods exist that are faster than Lin-Kernighan but these produce much lower quality solutions. Some construction methods need less execution time than Lin-Kernighan when producing solutions worse than 9% below optimal, see [Johnson 02]. When creating TSP solutions worse than 9% below optimal there are a variety of construction algorithms that are able to construct solutions very fast. For example Spacefill is able to generate solutions 32% below optimal in 0.06 seconds on a 15000 city problem [Johnson 02]. In contrast Lin-Kernighan achieves less than 2% below optimal in 40 seconds on the same problem size.

Lin-Kernighan uses partial solutions to guide its search for improvements. See Figure 4.1, Solution(iv) in Chapter 4 for an example of a Lin-Kernighan partial solution. A partial solution is created by, taking an existing complete solution,

removing one edge and inserting another. This partial solution is then used to create further partial solutions.

Lin-Kernighan only creates partial solutions that are one step away from a whole valid solution. This ensures that when the quality of the partial solution is estimated, the solution that is being used is almost valid. Any partial solutions can be converted into complete solutions by removing one edge and inserting another. This is what keeps partial solutions one step away from a whole solution.

Lin-Kernighan discards partial solutions that have a total distance that is more than that of the original complete solution. This is because all improvements can be located using only partial solutions that have a total distance less than the original complete solution. See [Applegate 99] for a description of the reason why.

Lin-Kernighan reduces the CPU time needed to find improvements by ignoring low quality partial solutions. Lin-Kernighan takes a whole solution and make a series of small changes to it. If any of the small changes result in a partial solution that is worse than the whole solution, the small change is treated as low quality and ignored. When it comes across a partial solution that is worse than the whole solution this is when it backtracks and makes a different small change. This works on the TSP because for any improvement there must exist a sequence of improving partial solution changes that locate the improvement; i.e. at no point does the sequence of small changes produce a partial solution that is longer than the original whole solution.

Lin-Kernighan is able to locate several improvements using a single pair of edges. After the first pair of edges has been changed and accepted it begins a process of looking for improvements. It will use the accepted high quality partial solution to locate further high quality partial solutions by changing pairs of edges and backtracking. All of the partial solutions are turned into whole solution to see if they result in an improvement. The best of these improvements is then implemented. [Applegate 99] shows that only accepting pairs of edges that improve the best running total so far can gain further speed; doing this had little effect on the final solution quality.

One of the aims of the experiments will be to compare methods of estimating the value of solution properties, where a 'solution property' for the TSP is an edge. A 'solution property' is being defined as a problem relationship that the move operator changes, e.g. TSP edges define how TSP cities relate to each other. Lin-Kernighan appears to be the only current method in the local search literature that estimates solution property values one property change at a time.

Lin-Kernighan works through the problem city by city looking for improvements. To avoid evaluating the same set of moves more than once, Lin-Kernighan marks cities that have not been evaluated and unmarks them after evaluation if only non-improving moves are found. When it makes an improving move the original Lin-Kernighan method re-marked all cities. [Bentley 90] used the idea of only re-marking cities involved in an improvement. This speeds up the search process, especially when there are a large number of cities.

Several methods of restricting the number of cities considered in a move are compared by [Applegate 99]. Restricting searches to the nearest 10 cities worked well for evenly distributed problems, but did not perform as well when cites were clustered. Out of the different methods of restricting the number of cities considered, Delaunay triangulation performed the best, see [Applegate 99].

Iterated Local Search is used as an effective way to improve Lin-Kernighan solution quality. There has been some confusion over the name, [Martin 91] created the method and called it "Large-Step Markov Chains", although Johnson called the method "Iterated Lin-Kernighan". The term "Chained Lin-Kernighan" now appears to have been adopted to describe this style of Lin-Kernighan hybrid, see [Johnson 02]. See section 3.1.6 for a description of Iterated Local Search.

Tour merging is reported to be an effective way of finding near optimal TSP solutions, useful in cases where improvements of a fraction of 1% are worthwhile, [Applegate 99], [Johnson 97]. TSP tour merging involves creating several solutions using Chained Lin-Kernighan (CLK). The next step is to look for improvements that can be made using solution properties that are different. To merge the tours [Applegate 99] used a Linear Programming based algorithm. This is slow compared to Chained Lin-Kernighan, around 2 seconds to reach 1% of optimal using CLK and 12 minutes using tour merging to find the optimal solution. These numbers are based on a 1500 city problem, see [Applegate 99] for a comparison of tour merging.

Relationship between Lin-Kernighan and Hypothesis. See Table 3.1

- Uses partial solution changes to guide the search for improvements
- Improvements are located using variable size move operators created using several partial solution changes.
- No preference is given to a particular initial partial solution but if several improvements are located using the partial solution, the largest is implemented.

3.5) Summary

The meta-heuristics described have many differences, but they also have two common threads. All the meta-heuristics described use the difference in solution quality between whole solutions to locate improvements and they use the difference in solution quality between whole solutions to choose which improvements to implement. Lin-Kernighan is not a general heuristic (meta-heuristic) and yet the evidence in later chapters shows that the Lin-Kernighan improvement location mechanism can be generalised and combined with different improvement preference mechanisms.

The methods described make changes to solution to help them find improvements. Tabu Search uses the difference in quality between whole solutions to work out which change to make next. Evolutionary Algorithms use the difference in quality between whole solutions to work out which solutions to merge and which solutions to discard. Ant Colonies use combined quality differences between whole solutions to select properties to build new solutions. Simulated Annealing uses the difference in quality between whole solutions to work out which non-improving changes to

make and which to ignore. Lin-Kernighan uses partial solutions to guide changes, other local search methods depend on whole solutions to guide changes.

Because there are several methods described and many variants of each it raises the question which one should be used with which problem. The thesis argues that distinguishing between location and preference will give a useful measure of their value.

Table 3.1 gives a broad overview of how most of the existing local search methods guide the search for improvements. The experiments in Chapter 5 show how location and preference can be compared. This allows the ideas that work well with a particular problem be identified. Table 3.1 gives a summary of the improvement location and preference methods used by the local search methods described in this chapter.

Meta-heuristics methods of guiding the search process

	Guide using	LK	TS	SA	EA	AC	SAGA	SAHC	SHC
Guidance objective: Improve improvement preference	Solutions of much worse quality than current best solution at **start** of search process			Y			Y		
	Solutions of much worse quality than current best solution at **end** of search process		Y						
	Large improvements first		Y					Y	
	Improvements that have a large move size at the **start** of search process			Y					
	Improvements that have a large move size at the **end** of search process		Y						
Guidance objective: Locate improvements	Individual whole non-improving solutions		Y	Y					
	Many whole non-improving solutions				Y	Y	Y		
	Partial solution changes	Y							
	Fixed size move operator(s)		Y	Y				Y	Y
	Variable size move operators	Y			Y	Y	Y		

LK – Lin-Kernighan
TS – Tabu Search
SA – Simulated Annealing
EA – Evolutionary Algorithm
AC – Ant Colonies
SAGA – Simulated Annealing Genetic Algorithm
SAHC – Steepest Ascent Hill Climber
SHC – Stochastic Hill Climber
Improve improvement preference – giving preference to using some improvements rather than others improves final solution quality.
Locate improvements – making changes to a solution helps locate improvements, the table lists different kinds of changes that are used.

Table 3.1 - Meta-heuristics – methods of guiding the search process

Comparison of performance of meta-heuristics on the TSP

Table 3.2 gives an overview of how well several of the different local search methods perform on the TSP. The methods in the table include many refinements and alterations designed to help them produce better TSP solutions. The table is a simplified view of how the methods perform on the TSP, for a more in depth examination of different execution times and instance sizes see [Johnson 02] and [Johnson 97]. The Simulated Annealing and Evolutionary Hybrid results are taken from [Johnson 97], the rest are from [Johnson 02].

10000 City Random Euclidean Instances from [Johnson 02]			
Local Search Heuristic		Excess Over Lower bound	Execution time
3-opt hill climber		2.88	25% faster than Lin-Kernighan
Lin-Kernighan		2.00	2.06 seconds
Tabu search		1.48	9141 times slower than Lin-Kernighan
Chained Lin-Kernighan		0.90	893 times slower than Lin-Kernighan
From [Johnson 97]			
1000 Cities	Simulated Annealing	1.6	2480 times slower than Lin-Kernighan on the same problem
431 Cities	Evolutionary and Lin-Kernighan Hybrid	-	With 35mins of execution time it was able to produce better quality solutions than Chained Lin-Kernighan on the same problem

Table 3.2 – Comparison of meta-heuristics performance on the TSP

4) New framework for location and preference mechanisms

Current local search methods locate improvements and give preference to using some improvements rather than others. At the moment it is difficult to see what impact each one is having on the solution. It is being argued that explicitly separating these issues will help us better understand which strategies work and why. The intention is to be able to use what is learned to improve existing local search heuristics.

All local search methods make use of previous solutions to find improvements. Because all local search methods use previous solutions it is argued that all improvement location methods can be described in terms of how they make use of previous solutions. This section describes how local search methods use previous solutions to locate improvements.

The methods of guiding the search for improvements that are being looked at in this thesis. There are some striking differences between how different local search heuristics guide the search for improvements. It is these differences that the thesis highlights and that the experiments compare.

Existing meta-heuristics and Lin-Kernighan are an example of this, they use different methods of locating improvements: -

- Existing local search meta-heuristics guide the search using the principle:

Take two whole solutions and use the difference in quality to estimate the value of the properties that are different.

- Lin-Kernighan [Lin 73] uses the following principle:

Use partial and whole solutions to estimate the value of solution properties that are different. This enables the method to estimate the value of solution properties one at a time.

Lin-Kernighan is a TSP heuristic, it is faster and produces higher quality solutions compared to methods based on the first principle. Yet none of the common meta-heuristics described in review of local search methods, chapter 3, use the second principle. A search of the local search literature failed to find any evidence that the second principle has ever been used on problems other than the TSP.

This case highlights the potential benefits of explicitly testing methods of locating improvements and improvement preference. Using partial solutions to locate improvements is just one method of locating improvements. Recognising this means we no longer treat it as a method only suitable for the TSP. The principle becomes just another method of locating improvements.

Explicitly testing methods of locating improvements and preference, on optimisation problems, will help us better understand the relationship between local search methods and problems.

4.1) Problem example and hypothesis

Below two different methods of locating improvements are highlighted. The example below is used to highlight the need to categorise and compare improvement location methods and improvement preference choices.

Most local search methods use differences between whole solutions to locate improvements. Partial solution changes appear to be used by Lin-Kernighan alone and it dominates the Travelling Salesman Problem (TSP). Figure 4.1 shows how whole and partial solutions are used to locate improvements.

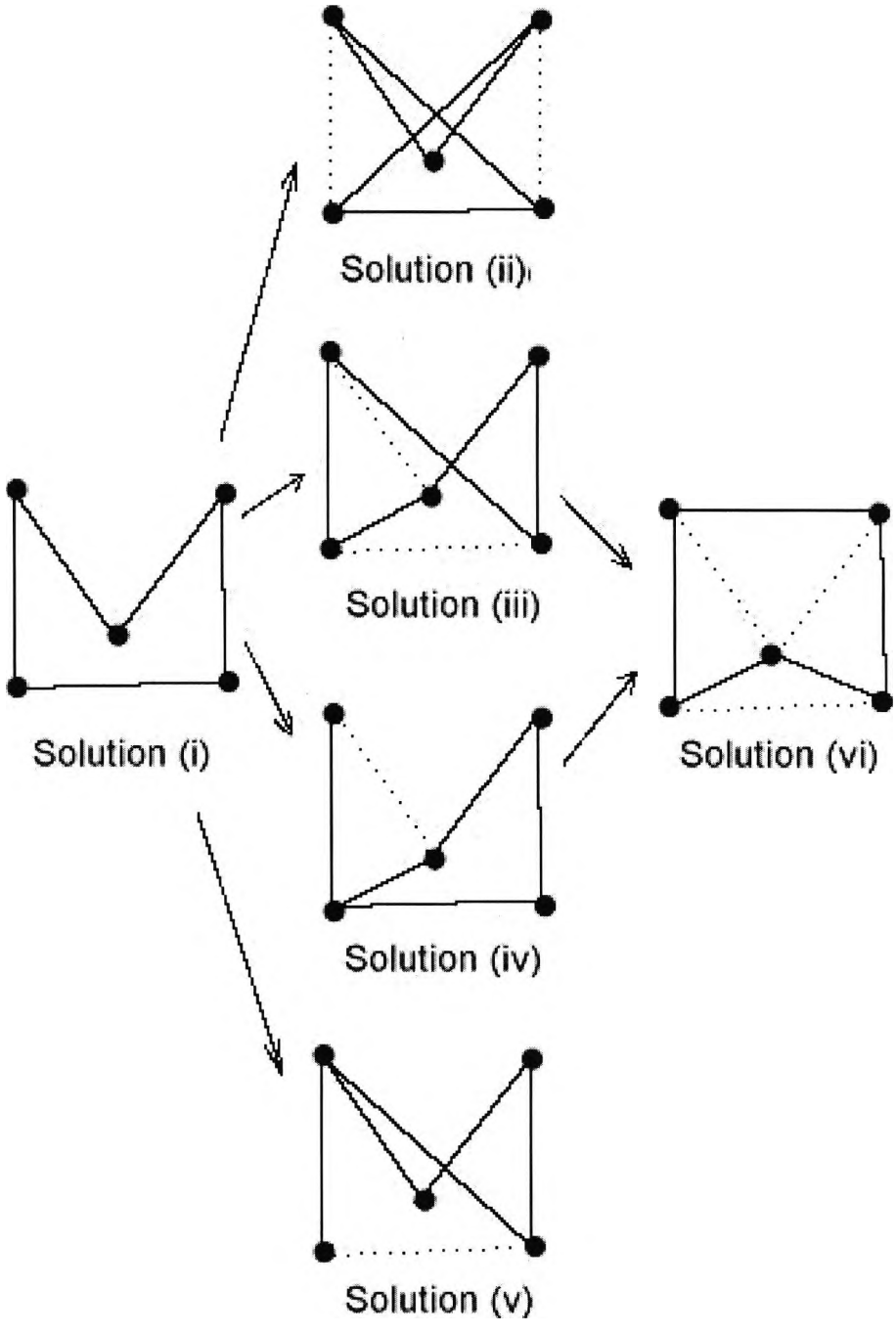


Figure 4.1 - Improvement location using partial and whole moves

Estimation of property values
Solution (i) is the current best, Solution (vi) is optimal,
Solutions (iv) and (v) are partial solutions

Figure 4.1 shows how the value of Travelling Salesman Problem properties (edges) can be estimated. Look at solutions (i) and (ii) in Figure 4.1, (ii) is much worse than

(i). This means the two properties, i.e. edges, that exist in (ii) and not in (i) are estimated to have a low value, local search methods would typically throw solution (ii) away. Most local search methods compare whole solutions such as (i), (ii) and (iii) to estimate the quality of solution properties that are different.

We can use estimates of the quality of individual properties to decide which properties to use to build new solutions. If we look at the difference between (i) and (iii), the solution quality is almost the same. This means the two edges that exist in (iii) and not in (i) are probably worth using to try and improve the current best solution. In this example the edge that goes from bottom left to middle is part of the optimal solution (vi).

When we use two whole solutions to estimate the value of properties, as is the usual case in local search, the same estimate is given to all the properties that are different. The two edges that exist in (iii) and not in (i) get classed as having equal value. Using only two solutions we have no way of guessing which of the edges has the most value.

When we look at solutions (iv) and (v), the partial solution change (v) has a better quality than (iv) and is also part of the optimal solution (vi). Using partial solutions gives a method of estimating which of the edges has the most value.

Estimates can potentially be improved by merging them together. By merging estimates from solution (i) (ii) and (iii), the edge from bottom left to middle gets a good estimate and it is part of the optimal solution (vi).

The fact that such a valuable method of locating improvements appears to exist in isolation highlights the potential gains of separating the issues of improvement location and improvement preference.

Hypothesis:

Distinguishing between improvement location and improvement preference is feasible and useful when creating local search algorithms.

4.2) Defining the problem concepts

This section defines the hypothesis issues and concepts in more detail. The descriptions link the hypothesis to the experimental design described in Chapter 5.

4.2.1) A Solution Property

Solution properties are what a move operator changes when it alters a solution. The move operator changes problem element relationships that need to be known to calculate fitness and check constraints. For example with a TSP an edge is a relationship between two cities. These relationships are the solution properties that the move operator changes and what we estimate the value of.

4.2.2) Locating improvements

We use solutions that are different from each other to estimate the value of the solution properties that are different. Estimates of the value of solution properties are used to try and improve the current best solution. Look at solution (i) in Figure 4.1, there are five edges we could add to it and five we could remove, the problem is how do we work out which to add and which to remove. We are able to estimate the

value of edges and use this to work out which removals and additions are probably part of an improvement.

The literature review describes several local search methods and how they locate improvements. These include using non-improving moves, partial moves and variable size move operators. See table 3.1 for summary.

4.2.3) Improvement Preference

Improvement preference is when preference is given to using some improvements rather than others. We have to make a choice of whether to implement the first improvement we are able to locate or give preference to particular improvements. A simple way of doing this is to give preference to large improvements in quality at the start of the search, rather than small improvements. As the heuristic progresses the preference for large improvements can gradually be reduced eventually implementing all improvements found. This is similar to acceptance criteria, although does not include moves that are rejected all the way through the search process.

Preference includes selecting one promising avenue rather than another. For example if there are two non-improving solutions and the method giving preference to using one of them rather than the other to try and locate an improvement.

The algorithms that evaluate preference measure the compound effect of improvement choices. The preference algorithms are not designed to assess the size of individual improvements or number of individual improvements.

4.2.4) High quality first & Low quality first

High quality first (steepest ascent) is implementing only large improvements at the start of the search. The solutions containing these large improvements are then used to locate further large improvements. This means large improvements are used to locate further improvements. At the end of the search when improvements become few and far between, the method implements all improvements, including improvements found using non-improving moves.

Low quality first (shallowest ascent) is the other way round. Where improvements located using non-improving moves are implemented at the start of the search. At the end of the search, again it implements all improvements.

This gives a sliding scale that can be used to locate improvements. At one end preference is given to low quality improvements located using non-improving moves and at the other end preference is given to high quality improvements located by high quality improvements. The experiments use such a sliding scale and the results are used to support this thesis.

A different approach is to give preference to improvements in the middle of this scale instead of giving preference to one end or the other. This involves giving preference to improvements in the middle of the range, and only allowing high and low quality improvements to be used at the end of the search. See the results chapter for a discussion on this.

4.2.5) Difference between location and preference

While methods of location find improvements they do not deliberately avoid or reject improvements. Preference is deliberately avoiding or rejecting improvements. Preference implements some improvements rather than others at the start of the search. At the end of the search all improvements found would typically be implemented.

Preference does not refer to any improvements that are rejected all the way through the search process. For example if non-improving solutions below some threshold are never used to locate improvements then these improvements could in theory be looked for but the method rejects them by never searching for them. Also, preference does not have anything to do with some improvements being easier to find than others.

4.2.6) Number of properties in an improvement

We want to improve the current solution, to do this we need to work out which properties of the solution to change. With whole solutions, the choice of which combination of solution properties we change is restricted by the fact we want whole solutions. When we use whole solutions we end up being forced to implement changes in the non-improving move that do not exist in the eventual improvement. With partial solutions we can assess which properties to add and remove as we build up a solution, back tracking as needed. This results in a change that is guided by the properties that make up the improvement. We use estimates of the value of properties to work out which properties to include and which to reject. Combining

estimates as we build solutions allows variable size move operators. This means we are deciding the number of changes as we build the move. [Lin 73] builds solutions in this way, evaluating as it builds. The number of properties in an improvement is another difference in how methods locate improvements and is a possible avenue of further research. The experiments in the thesis do not test this.

4.2.7) Number of properties in an estimate

Move size is the number of properties that we change to create a new solution. Using the difference between whole solutions to estimate the value of properties looks to cause a problem with choosing move size. The problem is balancing the objective of finding improvements, against calculating good estimates of property values. It looks as if the smallest move size gives the most reliable estimate of the value of a property. This is because the estimate is a combination of fewer properties. With a larger move size we assign the same estimate to more properties and so end up with more compound solutions to check. Equally, smallest move size is not always appropriate as improvements vary in size. Again, the experiments do not test this, this is being highlighted because it is another difference in how local search methods locate improvements.

4.2.8) Summary

Local search methods estimate the value of solution properties, the estimates are used to locate improvements and to guide the improvement preference. The speed and quality impact of different methods of locating improvements and deciding the improvement preference are measured.

4.3) Assumptions & hypothesis scope

The thesis only looks at local search methods and combinatorial problems, although the hypothesis may also apply to other methods and problems.

4.3.1) Local Search

The hypothesis focuses on local search methods. There are several names for the methods that the thesis focuses on, local search, meta-heuristcs and iterative improvement ([Zweben 90] and [Zweben 94] use the term iterative improvement).

4.3.2) Combinatorial optimization

The hypothesis only extends to combinatorial problems that have a finite number of possible solutions. Other styles of problem have not been looked at in depth. The hypothesis may or may not apply to other problems, but such problems are outside the scope of the thesis.

4.3.3) Locating improvements and preference

This thesis is arguing that the only things local search methods do is locate improvements and control the improvement preference. The thesis describes local search methods in terms of how they locate improvements and how they decide improvement preference. The thesis argues that explicitly separating these strategies will help identify which strategies work with which problem structures.

4.3.4) Construction

While it is expected that the hypothesis also apply to construction algorithms this is not examined in detail. Construction methods do locate new incomplete solutions and give preference to some incomplete solutions rather than others. Construction

methods and local search methods locate new solutions and give preference to some solutions rather than others because of this it is believed the hypothesis applies to construction as well as local search.

4.3.5) Solomon VRPTW Benchmarks

It is expected that the general aspects of the hypothesis expand to problems beyond the Vehicle Routing Problem with Time Windows (VRPTW) bench marks used in the experiments, see [Solomon 87] for the benchmarks. This is because the hypothesis aims to improve the matching of problems to meta-heuristics rather than a redesign of how they work. By separating improvement location from preference it is argued that our knowledge of which method suit which problems will improve. The no free lunch theory [Wolpert 95] suggests the need to create different meta-heuristics for different optimisation problems.

4.3.6) Using many whole solutions (crossover)

The experiments use the difference in quality between two solution to guide the search process and do not cover the use of many whole solutions. Because of this the thesis only argues on a theoretical level that the hypothesis also applies to using many whole solutions to locate improvements. The literature review describes how ants and GAs guide the search process using the difference in quality between many whole solutions and section 4.1 points out how estimates of quality can be merged to create a good estimate of solution quality. Further work is needed to better understand the impact of using many solutions to guide the search process.

4.3.7) Partial solution quality can be estimated

Problems where partial solution quality can not be estimated are not considered. That said, many greedy construction methods estimate the value of partial solutions when working out what to add next. Because of this, it appears possible to estimate partial solutions, with many combinatorial problems. Greedy construction methods exist for many of the well-known combinatorial problems, for TSP see [Johnson 97], for Scheduling see [Congram 98], for Time-Tabling see [Corne 96].

4.4) The gap in current knowledge

This section highlights knowledge gaps that appear to exist in the current local search literature.

4.4.1) Locating improvement and improvement preference

Current methods of local search treat the problems of locating improvements and guiding the improvement preference as one task. We can not therefore tell how well they perform at each. For example at the start of the search Simulated Annealing searches areas of the search space where the solution quality is much worse than the current best solution and at the end of its search it tends to avoid such areas.

It is hard to compare the preferences that Simulated Annealing uses with other methods of deciding what order to search the search space. It is difficult because Simulated Annealing intertwines how it locates improvements with improvement preference. They do not need to be intertwined, but intertwining them makes it hard to see what impact each strategy is having.

Tabu Search, searches in the opposite order to Simulated Annealing. Areas of the search space containing solutions a little worse that the current solution are searched before those where it is a lot worse.

Evolutionary Algorithms, Ant Colonies, Tabu Search and Simulated Annealing all make choices about the improvement preference and about where to look for improvements. The problem is they all merge both into a single task, making it difficult to compare which method is best at which objective.

Many meta-heuristic methods of locating improvements and guiding the improvement preference exist. Because there appears to be little agreement on which of the methods is best with which problem it suggests that we do not know which of them is best at which objective.

Explicitly separating these issues will help us better understand which strategies work and why.

4.4.2) Guidance using partial solutions

Current Local Search meta-heuristics use differences in the quality of whole solutions to guide the search for improvements. Lin-Kernighan [Lin 73] uses TSP partial solutions to guide the search and outperform all these methods.

Lin-Kernighan estimates the value of edges individually. Other heuristics estimate the value of several edges at once, assigning each edge the same estimate. These edges will typically be of different lengths and yet they all get given the same

estimate. It is suspected this is one reason why it dominates the TSP. Lin-Kernighan uses variable size move operators with only a single edge pair difference between partial solutions.

Lin-Kernighan is designed to work with the Travelling Salesman Problem, and dominates it, and yet variants of it do not appear to exist for other combinatorial problems. There appears to be a gap in the research for a general local search method that locates improvements using a variable size move operator, guided by partial solutions.

It looks as though little has been done to match methods of improvement location to problem structure. The fact that such a valuable method of locating improvements does not exist as a meta-heuristic shows important methods of locating improvements are being missed. The VRP and TSP are similar in structure and a search of the VRP and local search literature failed to reveal any evidence that the Lin-Kernighan method of locating improvements has been used on the VRPTW. This suggests little has been done to identify methods of locating improvements and match them to problem structure.

4.5) Reasons for merging local search methods

It is being argued that local search methods should be split into how they locate improvements and how they give preference to some improvements rather than others. It is being argued that separating these mechanisms will allow the impact of each to be better understood allowing location and preference mechanisms that suit the problem be identified and to be combined together.

Local search meta-heuristics appear to use only one depended variable to guide the search:

- difference in solution quality.

They also use several independent (choices) variables such as:

- move operator
- number of property differences between solutions
- improvement preference

All local search methods use these variables to guide the search. Evolutionary Algorithms, Ant Colonies, Tabu Search and Simulated Annealing all use these variables. This makes them look like ideal candidates for merging. Some work on merging Evolutionary Algorithms and Simulated Annealing has been done by [Poupaert 2001]. The method merges how convergence (preference) is guided.

Several different local search methods exist, there are many variations of each method and many hybrids. This gives us a large number of possible methods to pick from and a need for guidelines to help pick a method suited to the problem. The idea is to gather information about the strengths of different local search methods and then merge the strengths together to create new methods. Demonstrating the strengths of local search methods should allow the creation of methods better suited to the problems.

Tabu Search, Simulated Annealing, Ant Colonies and Evolutionary Algorithms all depend on using the difference in quality of whole solutions to estimate the value of properties. The experiments compare estimating property values using whole solutions with estimating property values using partial solutions, comparing speed and quality. With the aim of demonstrating which methods of estimating property values work best on the VRPTW.

Local search methods contain a huge variety of search control methods, move operators and problem styles. Because of this, the focus of the thesis is on merging guidance methods from well-known local search methods. The experiments will be restricted to the Vehicle Routing Problem. One reason for running experiments on the VRP is because it has multiple constraints. The multiple constraints of the VRP make it a much more realistic problem, as real world problems tend to have many constraints, see [Rochat 94] for an example.

By comparing methods of using the difference in solution quality to guide the search the results from the experiments show which methods work best with which problems.

4.5.1) Locating improvements using whole and part solutions

Current methods of locating improvements

Using a non-improving move to guide the search for an improvement can be seen as guiding a compound move. The change in solution quality of the first move is used

to guide the choice of which compound moves to evaluate. Figure 4.2 shows three non-improving guide moves, two of which are of acceptable guide quality, so only these two are used to create compound moves. A compound move is when one move is combined with another, this means instead of a large number of changes being made in a single move two smaller moves are used to achieve the same result. This is designed to save execution time, as the neighborhoods are much smaller so fewer evaluations are needed.

Local search methods use non-improving solutions in the search process. Non-improving solutions are used to locate solutions better than the current best solution. If we have a solution that is slightly worse than the current best solution, then there is a good chance the solution will have some properties that can be used to improve the current best solution. Typically if a solution is much worse than the current best solution than it has a low chance of having properties that can be used to create a solution better than the current best.

The more property differences that exist between solutions means there is a higher chance that one or more of the differences can improve the current best solution. The problem with having more differences is that there are more potential improvement combinations that need to be checked, as a result more differences may sometimes be a disadvantage.

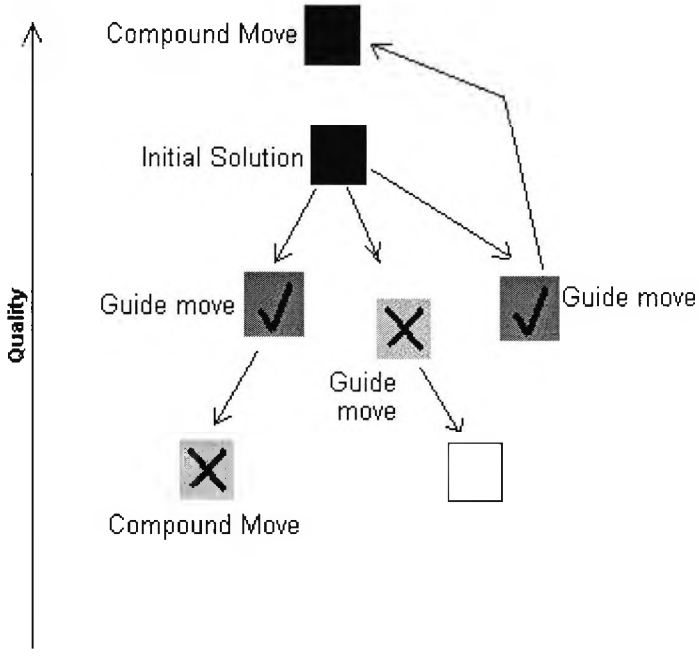


Figure 4.2 – Improvement location using non-improving moves

Simulated Annealing and Tabu Search make use of non-improving solutions that are typically very similar to the current best solution. EAs and ACs also make use of a neighbourhood of non-improving solutions. At the start of the search these solutions are typically quite different to the current best solution. It would appear this brings such methods much closer to an exhaustive search style approach.

Both Simulated Annealing and Tabu Search use the properties in non-improving moves to try and create better solutions. Simulated Annealing and Tabu Search make non-improving changes and use the reduction in quality to estimate the value of the properties changed. With Simulated Annealing and Tabu Search the estimate is typically discarded as soon as the move is made. Evolutionary Algorithms (EAs) and Ant Colonies are more complex, they can combine properties out of two or more

non-improving solutions to try and improve the current best solution. This allows them to combine several estimates together. For example EAs can take two non-improving solutions and use properties from each of them to improve the current best solution. With a non-improving solution, the lower the difference in quality the higher the estimated property value.

As noted in the literature review chapter, a population can be seen as the neighbourhood of the current best solution, very much like the neighbourhoods of Simulated Annealing and Tabu Search. Although the neighbors used in Evolutionary Algorithms and Ant Colonies tend to have more solution differences.

Evolutionary Algorithms and Ant Colonies use populations of solutions, and these solutions tend to be created randomly or by using greedy methods. For example, they start off by creating a population of random solutions. They then select pairs of solutions and then use the properties that are different to try and create an improved solution. The odds are the randomly created solutions will have a large number of properties that are different and so a large number of solutions will probably be needed to work out which of the property differences is part of an improvement.

When comparing two solutions each of the differences that assigned the same value. This suggests the larger the number of differences the less accurate the estimate of their value. Evolutionary Algorithms and Ant Colonies use property values to guide the search right from the very beginning of the search process when the number of differences between solutions is sometimes very high. It is suspected this hinders their ability to locate improvements in the early stages of the search.

Rather than discarding estimates of property values, Ant Colonies hold on to some estimates and merge them together. Evolutionary Algorithms are similar to Ant Colonies, in that they hold on to several whole solutions, rather than discarding them, thus holding on to information about the value of solution properties. It should be possible to test when holding on to property value estimates offers an advantage.

Proposed alternative method of locating improvements

It is proposed that Lin-Kernighan style of estimating the value of one property at a time is faster and more accurate, and this is one thing that will be tested. The Lin-Kernighan style of estimation has the advantage of being able to build solutions according to estimates, rather than building solutions to create estimates.

This thesis argues there is a need to test if partial moves are more effective at locating improvements than using whole move. Details of how the comparison between partial solution changes and whole moves was conducted are given in the experiments chapter, Chapter 5. At the moment there is little known about using partial moves to locate improvements. Comparing them with the performance of whole moves should help give an insight into when they are worth using.

4.5.2) Discarding solutions

Maintaining a population offers Evolutionary Algorithms a potential advantage over Tabu Search and Simulated Annealing, as Evolutionary Algorithms can hold on to solutions and continue to try and make use of them. Evolutionary Algorithms however have the problem of deciding if a solution is worth mutating or

compounding with another solution, if we are not going to do either we may as well discard it. Evolutionary Algorithms typically use differences in quality to decide which solutions to discard. Some EA methods also employ niching techniques, see [DeJong 75] and [Goldberg 87], where they keep hold of solutions that are different from each other. Because differences in solution quality and the number of property differences determine which solutions are discarded, I plan to measure how these variables affect performance.

Ant Colonies get round the problem of deciding what solutions to discard by maintaining a list of estimates of property values. Although as with Evolutionary Algorithms it is unclear at what stage the information becomes obsolete and no longer worth maintaining.

4.5.3) Improvement preference using whole and part solutions

Search control methods vary in the way they select which areas of the search space should be improved next. Search controls make choices about which changes to make when. As noted earlier Simulated Annealing searches the search space effectively in the opposite order to Tabu Search and it is not clear which order is better. As with finding solutions worth evaluating, existing search controls use the difference in quality between whole solutions to guide the improvement preference. In the same way that search controls use differences in quality of moves to decided what order the search space should be searched, we can use estimates of the value of individual solution properties to decide the order the search space should be searched. The experiments compare different methods of guiding the improvement

preference. I will be testing the use of partial changes to guide the preference as well as whole moves. Just as partial changes can be used to locate improvements they can be used to work out which improvements are given preference.

4.6) Summary

Local search methods change some properties of whole solutions to create better solutions. The properties we change are used to create estimates of their value. Estimates are used to decide which properties get used to locate improvements and guide the improvement preference. Most local search methods estimate solution property values by measuring the difference in quality between whole solutions and assign the same estimate to all the properties that are different. Because of the success of the Lin-Kernighan [Lin 73] it is being argued in this thesis, that the use of properties estimated on an individual basis, is likely to outperform the use of properties estimated with whole solutions, on combinatorial optimisation problems similar to the TSP such as the VRP.

5) Experiment Design and Pseudo Code

The experiments show how is it feasible to evaluate improvement location and preference. The experiment set-up describes how it is feasible to separate the two so the value of each can be assessed.

The experiments assess the value of using partial and whole solutions to locate improvements. They also evaluate the preference of using one improvement rather than another. The experiments are designed to compare location and preference methods with the aim of showing that distinguishing between the two is useful.

The experiments aim to show the strengths and weaknesses of methods of locating improvements and improvement preference. The literature review shows many methods that use both of these methods of guiding the search for improvements, but they are intertwined and treated as one method. The experiments show that by splitting them much can be learned about local search.

The two main experiments are as follows: -

- A hill climber to measure the speed and quality impact of using partial changes and whole moves to locate individual improvements to a solution.
- A hill climber to measure the speed and quality impact of giving preference to high quality changes and low quality changes.

The experiments use Solomon benchmarks [Solomon 87] for the Vehicle Routing Problem with Time Windows (VRPTW).

5.1) Designing the hypothesis, the experiments, and what I tried to learn

The aim here is to describe how I went about designing the hypothesis and the experiments. The section also describes what the experiments aimed to discover.

The major design choices were based on the following questions: -

- How much will be learned from the experiment?
- How long will it take to test the idea?
- Can the idea be implemented?
- Is the idea useful?

5.1.1) Creating the Hypothesis

Separately testing how improvements are located and the improvement preference appeared to be worth doing. The algorithms described in the literature review do work, what is less clear is why and how.

Steps and reasoning that led to the creation of the hypothesis

All local search methods make choices based on available information. In order to understand why and how they work it was necessary to understand what choices they make and what information they use to make those choices. The first step was to create a near comprehensive list of the information the algorithms use and choices they make.

The next step was to try to pin down which information was having a big impact. It was difficult to work out if a choice was having any impact on the end result and if

so, how much impact. The end result was what really mattered. Because of this it was assumed that every time an algorithm finds a better solution, a solution better than its previous best, it has produced something of value. The discovery of a single improvement, i.e. a new best solution, means something of value had been found.

Because new best solutions matter, it looked as if measuring how good an algorithm is at finding single improvement (new best solution) was worth doing. It is not about how long it takes to find many improvements. The aim was to discover how long it takes an algorithm to find a single improvement, i.e. how long different new best solutions take to find.

Next a list of different styles of improvement that can be found was created. These were as follows: -

- A simple 2-opt change, i.e. the smallest change that could be made that could result in an improvement.
- A 3-opt change that could be located using a 2-opt improvement, i.e. a 2-opt hill climber would locate it.
- A 3-opt change that could not be located using a 2-opt improvement.
- A 3-opt change that could not be located using a 2-opt improvement but there existed a 2-opt improvement that could be applied to the solution that would mean a 4-opt change would be needed to find the improvement.

It was assumed this was scalable, i.e. that a 2-opt change can be used to locate a 3-opt change and the 3-opt change can be used to locate a 4-opt improvement. This meant the idea could be scaled up all the way to a complete solution. This can be

done by starting with an initial solution and making several improvements to it. The total time it takes to find these improvements is the total time it takes to create the solution.

Working backwards it can be seen that local search methods use information to help locate these improvements. This information would take time to collect, in some cases a lot of time, and in others only a little. Also the quality of the information can vary, some information is going to be better at helping locate improvements than other kinds of information. Both these factors affect the ability of a method to locate improvements.

The second issue of improvement preference comes about because local search methods use one improvement, to locate further improvements. They have to choose which improvements are going to be used to locate further improvements. Preference could be measured on a small scale, looking at which 2-opt improvements are good at locating 3-opt improvements. But it is worth knowing the impact on a large scale. Because of this the experiments examine the compound effect of using one improvement to locate another to locate another, and so on. This is covered in more depth later in the chapter.

5.1.2) Theory verses experiments

I had a choice to make regards using theoretical analysis or experiments to test my theories. I decided to use experiments rather than doing a theoretical method of testing the hypothesis.

There were two reasons for this:

1. There is the problem of, what looks like it should work in theory, does not always work on the ground when implemented. This is summed up by Smith [Smith 04]

   > "Usually, to make something amenable to mathematical analysis it is necessary to make some simplifying assumptions, and that often makes the thing unrealistic."

   Because of this it was decided that implementing an actual optimiser would mean the results would be more representative of how people actually go about solving combinatorial problems.

2. Second I wanted to go beyond the TSP. I wanted a more complex problem to test my ideas on. Real world problems tend to have many complicating factors and I wanted a problem that was closer to a typical real world problem. I felt modelling a complex problem such as the VRP would be very difficult to do analytically. For these reasons, and others, the Vehicle Routing Problem (VRP) was used to test the hypothesis.

5.1.3) The 5-opt hill climber

I wanted to be able to compare the performance of using different methods of locating improvements. One of the major problems was to compare using partial changes with using whole moves. The solution that I came up with was to use what in essence is a 5-opt hill climber. I created experiments in pairs so I could compare the impact on performance of the difference. I tried to keep both experiments similar to each other and keep them close to how they would be implemented in reality. All

pairs of experiments were able to locate and implement 2, 3, 4 and 5-opt improvements. They were able to locate all of these improvements in the available neighbourhood. The way they located improvements and the choices of which improvements got implemented were compared.

Comparing the ability to locate beyond 5-opt size improvements was considered but was not implemented. The amount of execution time needed had the potential to sky rocket. This would have meant limiting the methods in some way to ensure they could run in a reasonable time. I also looked at allowing the partial change hill climbers to be able to locate 4-opt improvements that would have been very hard for the whole move method to locate. In the end I decided not to do this. This is because it made the coding of the partial change method simpler and the experiments would be the same in terms of which improvements they could locate.

5.1.4) Time taken to locate and improvement

The algorithms that the experiments compare are in essence 5-opt hill climbers. This means they are able to all locate more or less the same improvements and thus allow the time taken to locate these improvements to be evaluated.

It should be noted that a few of the improvements in the neighbourhood are missed because of the way the algorithms work. The methods use solution changes to locate improvements and there are some low quality changes that are not used to locate improvements. This is because there is only a small chance the change will be able to help locate an improvement.

In order to test if one method of locating improvements is better at locating improvements than another, several experiments were done that continually recorded execution time and solution quality. Several types of information exist that can be used to locate improvements. The experiments examine four approaches: -

- Difference in distance
- Arrival time at the customer
- Two styles of combining the difference in distance and arrival time

Partial change and whole solutions are also used to locate improvements. Partial changes and whole solutions are combined with the other four, giving a total of 8 different methods of locating improvements. These are described in chapter 7, the results chapter.

5.1.5) Compounded whole 3-opt move

Existing meta-heuristic local search methods use whole moves, i.e. complete solutions, to locate further whole moves. The experiments are designed to learn how good whole moves are at locating improvements. The experiments used 3-opt changes to locate improvements. I designed the experiments so they could use both improving and non-improving 3-opt moves to locate 4 and 5-opt improvements. The 3-opt move could also create 2-opt moves by removing and then re-inserting the same edge in a single 3-opt move. The partial change method could also do this as it meant coding was less complex. I wanted to know the impact of using different quality whole 3-opt moves to locate improvements. I designed the code so I could vary the quality of the 3-opt moves used to locate improvements. The aim was to be able to directly compare different methods of guiding the search for improvements.

5.1.6) Compounded partial solution change

Partial change information can also be used to locate improvements. The partial move operator uses a single edge as a starting point. If changing the edge looks promising it will use it to build 2, 4 and 5 opt moves. Each time it changes an edge to increase the size of the potential move it evaluates the change to see if it still looks promising. As with whole moves the code is designed so the quality of the edges used to locate improvements can be varied.

5.1.7) Picking the best improvement

The code makes choices about which improvements to implement and which to ignore. This gives rise to probably the biggest difference between the partial change and whole move comparisons. Because a single 3-opt whole move can sometimes locate several compound improvements, I need to decide which of them to implement. The same goes for part solution changes, one edge change can sometimes locate several improvements. The problem is the improvements they will locate will not be the same. This means the choice of which improvements are picked to be implemented are not going to be worked out in the same way. I can still control the quality of the change used to locate these improvements which makes them similar. The problem is a set of improvements located by one method is not picked in the same way as a set of improvements located by the other method. While ways round the problem exist I saw little point in solving it. The way I implemented it made coding simpler and wasted less execution time so was closer to how you would approach a real world problem.

5.1.8) Control of time quality trade off

As noted earlier, we want to find better quality solutions and in less time. Execution time and solution quality are both important. This means I would like to be able to alter the amount of time it takes the algorithms to run. I want to do this even if it means I end up with a worse quality solution. Because I can control the quality of change used to locate improvements I should also be able to control execution time. The experiments have been designed to show how much control of execution time can be gained by using different quality changes to guide the search.

Solution quality and execution time are continually recorded and afterward these were split into three groups for analysis. The groups are created based on execution time so the strengths and weaknesses of the different methods can be analysed. The 3 groups are:

- Fast early solutions – for problems when a short execution time is better
- Point at which early solutions are overtaken – point at which to switch algorithms
- Best quality solutions – for problems when longer execution times are acceptable

5.1.9) Preference design

The experiments look at giving preference to using a low quality change to find an improvement verses using a high quality change. Because local search methods use improvements to locate further improvements they sometimes give preference to using one type of improvement rather than another to locate further improvements. This preference is based on the quality of the change used to locate the improvement and also the quality of the improvement itself. The experiments are designed to show whether using low quality changes to locate improvements is better than using high quality changes. Because I can control the quality of change that gets used to

locate an improvement I can use this to alter the kinds of improvements that are used to locate further improvements. The code is also designed to let me control which improvements get implemented. We use a small change to locate a set of several improvements and we then implement one of them. The code is designed to pick which of the improvements in the set gets implemented. It can pick either the best improvement or the worst improvement in the set. This allows me to control if low or high quality improvements get used to find further improvements. The experiments have been designed like this so I can measure the impact of using different kinds of improvements to locate further improvements.

5.1.10) Linking experiments back to hypothesis

The thesis argues that existing local search meta-heuristics use the difference in quality between two whole solutions to locate improvements. It is the use of this difference between solutions to find improvements that has been examined. The experiments do not compare performance against methods such at Tabu Search or Simulated Annealing. Because meta-heuristics use whole solutions to locate improvements it is argued the results offer an insight into how solution changes can be better used to locate improvements.

The experiments are designed to assess the value of using partial changes and whole solution changes to guide the search. This is done on two levels, first the ability to locate improvements is tested, second the improvement preference is compared.

The experiments test several aspects of the hypothesis. The experiments compare using partial changes against using whole moves. To test improvement location and

improvement preference, two different sets of experiments were created, both of which compare using partial changes and whole moves. The first set of experiments test improvement location methods, the second compare improvement preference.

5.1.11) Preference testing

The experiments are designed to test which kind of improvement preference is better and also test if partial changes are better at guiding preference than whole solution changes. This is tested using the VRP. Because the VRP with tight time windows is similar to a scheduling problem and a VRP with slack time windows is similar to a TSP it is expected the experimental preference results will generalise to these similar problems, although the experiments described here do not test this.

5.2) Methodology - Data Sets, Speed and Number of Runs

The reliability and general usefulness of the results is dependent on variables such as the choice of data sets used and the number of execution runs etc. This section describes how the experiment set-up aims to maximise the usefulness of the results.

5.2.1) Data Sets

With combinatorial problems a small increase in the number of elements can produce a large increase in the number of possible solutions. This makes it important to understand the impact of different sizes of problem on speed and solution quality. This is because of the potentially large variation in the size of the solution search space. Two different problem sizes were used in the experiments, 100 customers and 200 customers, see tables 5.1 and 5.2. This allowed the experiments to show how scaleable the different methods are.

VRPTW Data Set Descriptions, Large Time Windows					
Name of Data Set	C2_1	Rc2_1	Rc2_2	R2_1	R2_2
Number of Customers	100	100	200	100	200
Distribution of Customers	Clustered Customers	Mix of Random and Clustered Customers	Mix of Random and Clustered Customers	Randomly Distributed Customers	Randomly Distributed Customers

Table 5.1 – VRPTW Data Sets - Solomon and Extended Solomon

VRPTW Data Set Descriptions, Small Time Windows						
Name of Data Set	C1_1	C1_2	Rc1_1	Rc1_2	R1_1	R1_2
Number of Customers	100	200	100	200	100	200
Distribution of Customers	Clustered Customers	Clustered Customers	Mix of Random and Clustered Customers	Mix of Random and Clustered Customers	Randomly Distributed Customers	Randomly Distributed Customers

Table 5.2 – VRPTW Data Set Descriptions - Solomon and Extended Solomon

For several of the 11 VRP problem types, only 8 benchmark instances were available. When 10 benchmark instances were available, only the first 8 of these instances were used. The algorithms were evaluated using the 11 different VRP problem types listed in tables 5.1 & 5.2. The Solomon benchmark problems, [Solomon 87], contain two different types of time window constraint and three different customer distribution styles. The small time window instances are short haul problems because fewer customers can be fitted into a route. Whereas the large time window instances longer haul problems because they can contain more customers per route. The clustered instances mimic towns and cities because the customers cluster together. The instances that contain a mix of random and clustered customers tend to be closer to real world VRP problems because not all the

customers are clustered. The other style of distribution is simply a random distribution of customers.

5.2.2) Speed

The experiments measure execution time against solution quality. The experiments also test how well the methods scale as problem size increases. To test how well methods scale, two different problem sizes will be used. While the methods that find the best quality solutions are of interest, the focus is also on measuring the ratio between speed and solution quality for different problem sizes. The experiments record the solution quality and execution time as the search progresses, this allows performance in the early and late stages of the search to be compared.

The computer used was an Intel P3 MMX 450Mhz with 192 MB of RAM running NT 4. The experiments were implemented in Java and executed using Java 1.3.0.

5.2.3) Number of Runs

Local search methods are often stochastic therefore the more runs that are performed on each problem instance, the more reliable the average and standard deviation figures become. Although this is irrelevant with deterministic methods as they produce the same result every time.

Each algorithm was tested on each problem style 40 times. Both the average and standard deviation were recorded, but the results for individual runs were not stored, see appendix for standard deviations. The 40 runs comprise of 5 runs on each of the

8 different benchmark instances. This means the experiments use 40 runs per method type.

5.3) Initial Solutions

The aim is to create several different solutions that can be used for all the experiments. A fairer comparison can be achieved if the experiments being compared use exactly the same starting solutions. A set of random initial solutions is created, and this set is used for all the experiments. Random solutions were used rather than greedy solutions because I wanted to test the ability of different algorithms to deal with a more diverse set of starting points.

Algorithms are compared by comparing how much they managed to improve the initial solution. All the methods use the same initial solutions, they all start from exactly the same place. The percentage improvement in relation to initial solution quality is used to compare algorithms. Both solution quality and execution time are important and the trade off between time and quality can be used to help select an algorithm. This means to measure the increase in time a fixed point is needed. Because the lower bounds do not offer a fixed point that can be used to measure the increase in execution time, both execution time and solution quality are measured using the completed random initial solution as a starting point.

A different set of initial solutions was created because I wanted to know about how the problem changed as solution quality improved. Starting with the random initial solutions and using a hill climber created these solutions. I used a hill climber to create several initial solutions, at various stages of the hill climb sample solutions

were collected and these were used as initial solutions. The aim being to take these solutions and look at how easily improvements could be found. Running the hill climber twice picked out the sample solutions. The first run is used to get the proportion of improvements to non-improvements found in a complete run. The second run uses the proportion of improvements to non-improvements to work out what stages to collect the solutions from. The second run saved a copy of the solution every one fifth of the way through the hill climb, see Figure 5.1. These solutions were then used by the threshold calculation experiments. The threshold calculation experiments assess how easy it is to find improvements.

```
/*
Basic Hill climber
    Creates an initial solution and makes a series of improvements to it.
    Used to gather set of initial solutions for main experiments.
*/

create an initial random solution

DO WHILE untested small solution changes exist
    make a small untried change to the current best solution

    IF the small change produces an improvement
        IF proportion of improvements to non-improvements > next save point
            save the solution to file
            set next save point so it saves at regular intervals
        END IF
        make note of new best solution
    END IF

END DO
```

Figure 5.1 - Creation of initial solutions

5.4) Finding improvements – Threshold Calculation

Local search methods create solutions to help find improvements. The aim here is to find out the odds that a particular pair of solutions can be used to find an improvement. The method takes an existing solution, changes it and compares the quality of the two solutions. The next step is to establish if some parts of the change can be used to create an improvement. The first small change is checked to see if it can be used to create a 4 or 5-opt improvement. If the small change can be used to locate a 4 or 5-opt improvement then the small change quality is recorded so the average and standard deviation can be calculated. The method calculates the average and standard deviation of all the smaller changes that can be used to help locate 4 and 5-opt improvements. The change in solution quality of the smaller change can now be used to locate 5-opt improvement. We can use the resulting average and standard deviation to work out the odds that a small change will help us locate a 4 or 5-opt improvement. See Figure 5.2

We want to be able to locate improvements when the solution quality is low and when it is high. We want to be able to locate 4 and 5-opt improvements using small changes at every stage of the search process. Different quality solutions are used to calculate the average and standard deviation of using small changes to locate improvements. Averages and standard deviations are calculated for the various stages of the hill climb. The initial solution created earlier by the hill climber are selected based on the proportion of improvements to non-improvements found and are used to calculate the averages and standard deviations.

I want to compare the ability of partial changes and whole moves to locate improvements. Both partial changes and whole moves are able to find all of the 5 opt improvements in the neighbourhood. By not bothering looking for the hard to find 5-opt improvements we can normally speed up the search with only a small impact on solution quality. These experiments calculate averages and standard deviations so I can measure at what point improvements become 'hard to find'. The threshold calculation experiments assess the ability of 1, 2, 3 and 4-opt partial change to locate improvements and the ability of 2 and 3-opt moves to locate improvements.

```
/*
Locate improvements – Threshold Calculation
The aim is to work out which small changes can be used to locate
improvements. It works out the odds of 1, 2 & 3-opt changes being able to
locate 4 & 5-opt improvements.
Take a single solution and for each type of guidance change that can locate
an improvement calculate average and standard deviation of the guidance
changes.
Do this with the different quality solutions collected from the hill climb.
Note distance and time window slack time are the two estimates of change
quality being used.
*/


DO FOR each initial solution of similar quality collected hill climber
    DO
        make a compound 4 or 5-opt change to initial solution
        IF improvement
            store quality of 1, 2 and 3-opt changes that located the
improvement
        END IF
    WHILE more sample compound moves required

    calculate average and standard deviation of each 1, 2 and 3-opt change
        that located an improvement.
END DO
```

Figure 5.2 - Find improvements to a single solution

5.5) Finding improvements – Percentile tail off point

These experiments are designed to find the quality cut off, below which it is not worth checking to see if a small change is part of an improvement. These experiments gradually lower the quality of the small changes that are used to locate improvements. This is done until improvements in final solution quality tail off. By changing which solution changes are used to locate improvements we are able to adjust how many improvements get found. The experiments are run in sequence starting with a 5% threshold that is able to locate 5% of improvements. A 5% threshold means the algorithm will avoid evaluating large numbers of non-improving moves. The algorithm gradually increase the threshold to 95%, this is able to locate 95% of improvements but also evaluates large numbers of non-improving moves. The tests are run using a range of percentiles calculated using average and standard deviation from the previous experiments. The percentile intervals used are 5%, 10%, 20%, 30% ... 80% 90%, 95%. The experiments run hill climbers to try to discover the optimal percentile cut off point, see Figure 5.3. The percentile at which the improvement in solution quality falls below 0.5% is taken to be the optimal percentile cut off point.

Altering which changes are used to locate improvement affects solution quality and execution time. These experiments are designed to measure the trade off between solution quality and execution time. I want to know how the time quality trade off is affected by using different quality small changes to locate improvements.

We can locate a single improvement using one of many different small solution changes. The lower quality small solution changes have a lower chance of being part of an improvement. The odds are that I can ignore the lowest quality changes and still find all the 5-opt improvements. Even though some of the low quality changes could be used to locate 5-opt improvements, better quality changes can normally be found to locate the same improvements. Because most low quality changes do not result in the location of an improvement, we save valuable execution time by ignoring these.

The experiments test if partial changes are more effective at locating improvements than whole moves. The experiments are designed to show which is more effective in terms of speed and quality. The aim is to show not only if partial changes are better at locating improvements, but also to show which kind of information is better at locating improvements on the VRP. I used the difference in distance, the arrival time at the customer and a combination of the two, to locate improvements. I wanted to get a feel for different kinds of information that can be used with partial changes and whole moves.

```
/*
Locate improvements – Percentile tail off point
The aim is to work out which small changes are worth using to locate
improvements. The experiment is designed to find the cut off point, below
which it is not worth checking to see if the change is part of an improvement.
*/

DO FOR each small change percentile, calculated using average and
        standard deviation.
    DO FOR each random initial solution
        DO
            make a small change to the current best solution
            IF small change is better quality than small change percentile
                IF we can use the small change to locate an improvement
                    implement the improvement, becomes best solution
                END IF
            END IF
        WHILE number of consecutive rejected small changes is still small
    END DO
    calc average & standard deviation solution quality at several time
intervals

END DO
Find percentile where improvement in final solution quality falls
    below 0.5 percent
```

Figure 5.3 - Find improvements to a single solution

5.6) Guide improvement preference

Local search methods use improvements to help locate further improvements. When an improvement is located, it does not have to be used to locate further improvements, instead preference can be given to using other smaller or larger improvements to locate more improvements. The same applies to non-improving moves, we can give preference to using small or large non-improving moves to locate improvements. The experiments aim to show whether using low quality changes to locate improvements produces better quality improvements than using

high quality changes. The question is when should preference be given to using large improvements to locate further improvements and when should preference be given to non-improving moves and small improvements. The experiments also look at the impact of giving no preference at all and so using every improvement found to locate further improvements.

The main two experiments are low quality changes first and high quality changes first. *Low quality changes first* is tested by allowing only the lowest 5% of solution changes to be used to locate improvements at the start of the hill climb. Eventually the number of improvements we are able to find tails off. At that point the percentile is increased so that slightly higher quality changes get used to find improvements. This results in only low quality changes being used to find and implement improvements at the start of the search process and leaving the high quality changes till last. *High quality changes first* work the other way round.

The aim is to see if using a particular preference offers an advantage in terms of the time quality trade off. The experiments examine if solution quality can be improved by using one improvement rather than another to locate further improvements.

These experiments are roughly based on Simulated Annealing and Tabu Search methods of move selection as described earlier. The experiments use hill climbers to evaluate different methods of working out what to improve next.

The experiment examines the progress of the hill climber as it finds and implements improvements. It records the solution quality, execution time and the percentile

acceptance threshold as the hill climb progresses. These are used to create an average and standard deviation of the hill climber's performance.

```
/*
Improvement preference
Test if final solution quality is affected by altering the improvement
preference.

Two hill climber preference types are tested: -
    Uses small changes of low quality to begin with then gradually expanding
        to all.
    Uses small changes of high quality to begin with then gradually
        expanding to all.

*/

create two lists of small change acceptance thresholds,
        one increasing, one decreasing.

DO FOR increasing then decreasing acceptance thresholds
    DO FOR each random initial solution
        set initial acceptance threshold to first threshold in list
        DO WHILE number of consecutive rejected small changes is still
small
            make a small change to the current best solution
            IF small change is less than upper acceptance threshold AND
                    greater than lower acceptance threshold
                IF we can use the small change to locate an improvement
                    implement the improvement, becomes best solution
                END IF
            ELSE
                IF the number of rejected small changes is too big
                    set acceptance threshold to next threshold in list
            END IF
        END WHILE
    END FOR
    calc average & standard deviation solution quality at several time
intervals
END FOR
```

Figure 5.4 - Find improvements to a single solution

5.7) Move operators and combining problem styles.

The following covers some of the issues I took into account when designing the operators.

Combinatorial problems often have multiple objectives and constraints, this results in them being made up of a mix of problem styles. For example take a Vehicle Routing Problem with capacity constraints, both customer adjacency and the vehicle assigned affect the solution. This causes complications with the move operator as changes to both adjacent position and absolute position need to be taken into account when a change is made. For more information on problem styles, see section 3.3 Move Operator Design.

5.7.1) Move operator restrictions

Move operators are restricted to limit which solutions they consider. This appears to be done in two ways: -

- Restricting move size – this controls how similar new solutions are to an existing solution.
- Restrictions on legal moves – methods sometimes remove unpromising properties from the search, such as Granular Tabu Search for the Vehicle Routing Problem, which removes the longer edges [Cordeau 02].

Changing the restrictions on the move operator can change the number of evaluations needed to find an improvement. Move operator restrictions affect the

proportion of improvements available in the neighbourhood, and the average improvement size.

5.7.2) Move operator used

A 3-opt shift operator is used to try to maximise the proportion of improvements found. The 3-opt shift operator moves a customer from one place in the route to another. The move operator could move a customer to a different position within the route or to another route. This style of move operator was picked because it allowed the arrival time at a customer on a route to remain more or less the same. When a customer is inserted into a route, only the customers after the insertion point are affected. Their increase in arrival time is dependent on which customer is inserted.

I used a compounded 3-opt shift operator to find 5-opt improvements. The maximum move size used was a 5-opt. While the code was designed to allow any number of 3-opt moves to be compounded I prevented it from going beyond a 5-opt. This was done in order to keep the experiments simple, move operator size was an extra variable I decided not to test.

Both the partial and whole move methods used a compounded 3-opt shift operator. I wanted the experiments to have very few differences. While partial moves are capable of complex 5-opt moves that do not disrupt arrival times very much, I decided to limit both partial and whole moves to being the same. This made implementation simpler and allowed more of a like-for-like comparison.

The 3-opt move operator allowed a 2-opt to be created and the compounded 3-opt allowed 4 and 5-opt moves to be created. This was a simple matter of not checking to see if the edge being inserted or removed was already part of the move. This meant that extra code did not have to be created to create 2-opt and 4-opt moves. The down side to this was that the estimate of the moves value was less accurate. The loss in accuracy is because the edge was being used to create the estimate but was not being used in the move.

5.8) Summary

The experiments are designed to compare the performance of improvement location and preference mechanisms in terms of solution quality and execution time. The experiments test the ability of non-improving solutions and partial solutions to locate improvements and guide the improvement preference. Speed and solution quality is measured for different solution sizes.

6) Results of Formative Experiments

Several design choices were made based on the results of formative experiments. The formative experiments were designed to try to discover what did and did not matter when detecting improvements and changing the improvement preference. The formative experiments saved samples of several neighbourhoods so I could manipulate them in a spreadsheet to try and work out what mattered. I saved details of all the 2, 3, 4 and 5-opt improvements in the neighbourhood along with details of what needed changing to implement the improvements.

The results in this chapter were used to guide the design of the main experiments. The main experiments make heavy use of acceptance thresholds, for both improvement location and improvement preference. These results described in sections 6.2 to 6.4 guided the choice of initial acceptance thresholds and how they changed. The results described here impacted sections 5.4, 5.5 & 5.6. Section 5.4 describes how the acceptance threshold percentiles were calculated by sampling the neighbourhood, section 5.5 describes how the lower percentile cut off point was picked and section 5.6 describes how the preference algorithms changed the thresholds as the search progressed.

In relation to the hypothesis, the results described in this chapter indicate that turning off the preference mechanism is a useful method of tuning the acceptance thresholds.

In most cases the results described in this chapter are based on a small number of cases. To assume the results described here apply to other instances of the same

VRP problem is risky. The formative results may help in our understanding of local search and the VRP, because of this I described them here. I would recommend further experiments before making extensive use of the formative results.

6.1) Saving the neighbourhood at different stages of the hill climb

To collect the neighbourhood details hill climber algorithms were used. At various stages in the climb, information about the solution and neighbourhood were collected for analysis, see section 5.3 "Initial Solutions" for experiment description and pseudo code. A 3-opt hill climber was executed twice. The first run was used to find out how a typical run progressed. The second run saved details of all the improvement in the neighbourhood at various stages in the search. To work out at what points to save the neighbourhood several pieces of information about the hill climb were saved: -

- Number of improvements found.
- Number of non-improvements found.
- Total improved distance since initial solution.
- Total time sat waiting for customer time windows to open.
- For every non-improvement, total of all increases in distance.
- For every non-improvement, total of all time window violations.
- For every non-improvement, total of all improved distances with invalid time windows.
- For every non-improvement, total of all wait times with non-improving distance.

These metrics (variables) were considered when searching for a method for measuring the progress of the search. The method that was eventually picked was the proportion of improvements to non-improvements. At the start of the search, the hill climber was able to find a lot of improvements, as the search progressed, improvements became harder and harder to find. This meant that simply counting the number of improvements and non-improvements found gave quite a reliable method of monitoring progress.

In order to calculate acceptance thresholds, a sample of improving and non-improving move details was saved for each hill climb stage. There were too many neighbours in a neighbourhood to save all of them. All of the non-improving moves would not fit in the spread sheet and they took time to save so I decided to just save a sub set. I also did the same with improving moves for some of the larger problems.

6.2) Non-changing threshold

As the hill climb progressed it was observed that the thresholds for locating improvements did not change. The methods use small changes to locate improvements. The quality of the small changes needed to locate say 70% of the improvements in the neighbourhood did not change as the hill climb progressed, see section 5.4 "Finding Improvements – Threshold Calculation" for details of how acceptance threshold information was collected.

The non-changing threshold observation is based on the results from 3 large time window problems with clustered customers. Each of these was executed twice, giving 6 samples to examine. The graph, see Figure 6.1, contains results from 6

separate stages of the hill climb. The results did not show any patterns that indicated one hill climb stage was much different than another.

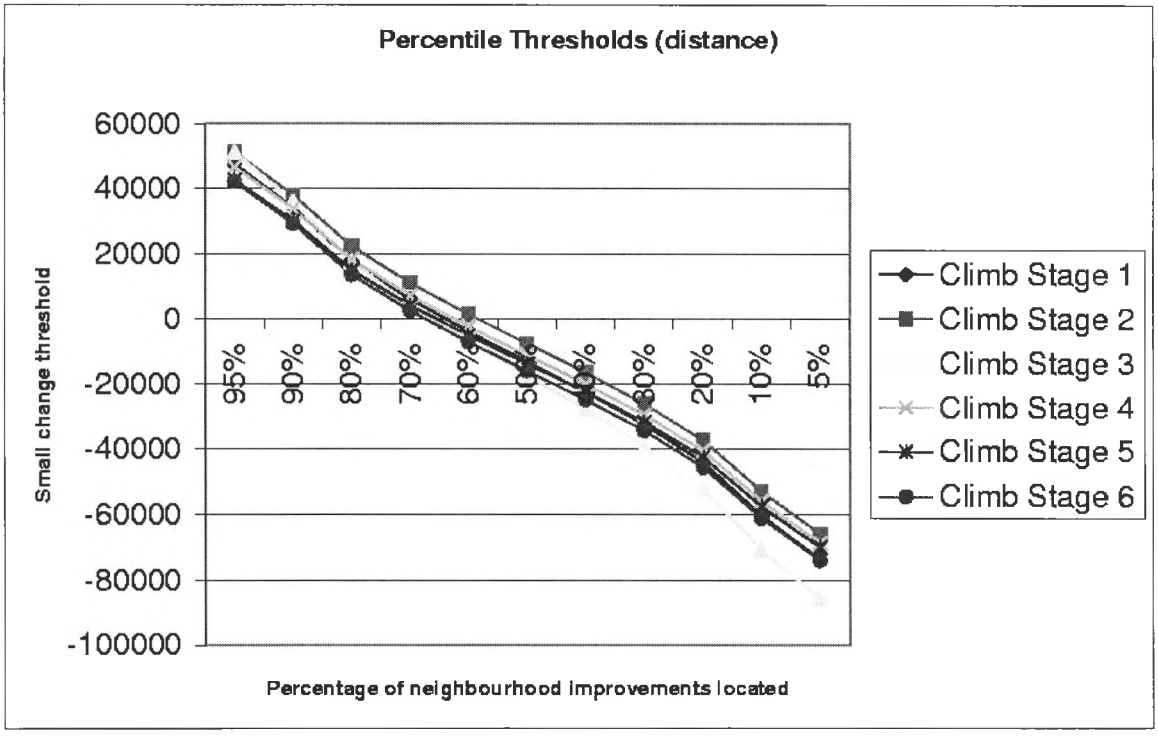


Figure 6.1 - Percentile thresholds used to locate 5-opt improvements

As a result of this set of formative experiments the decision was taken to use the same set of threshold levels all the way through the search in both of the main location and preference experiments, see sections 5.5 and 5.6. The original plan was to collect separate thresholds for the various stages of the search. The idea behind collecting thresholds at various stages was to have better control of performance by being able to match thresholds to the current neighbourhood. Because improvements did not appear to get any easier or harder to find as the search progressed, the decision was taken to drop the idea of collecting thresholds for the various stages of the search. This means a set of thresholds was calculated before each hill climb experiment and that set of thresholds was used for the entire hill climb.

The results in figure 6.1 are for large time window problems with clustered customers. The results consisted of a sample population of 6 and results for 6 stages of each hill climb for each sample. This gave 36 results, all of which were very similar.

The non-changing threshold suggests that improvement location is unaffected by solution quality. This suggests that by turning off the improvement preference mechanism an accurate acceptance threshold can be found, i.e. it allows us to work out which non-improving solutions can be used to locate improvements. This is useful because it allows the improvement location mechanism to be tuned before a preference mechanism is applied.

6.3) Setting the improvement location thresholds

This section describes issues that relate to the setting of an improvement location threshold. Lin & Kernighan, [Lin 73], point out that different small changes can locate the same improvement. This means that sometimes there is more than one promising 3-opt move that can be used to locate a 5-opt improvement. This means some of the 3-opt changes that can be used to locate improvement can be ignored and it is still possible to locate all the improvements in the neighbourhood. The formative experiments were used to get a feel for how effective this idea was. The formative experiments showed that the best 90% of the partial changes used to locate improvements could be used to locate all the improvements in the neighbourhood. See section 5.3 "Initial Solutions" for details of how the neighbourhood information was collected. Even lowering the threshold to 80% made

little difference. The formative experiments suggested that using a threshold that was 10% below the optimal would make little difference to the final solution quality. This suggested intervals of 10% could be used to locate the near optimal threshold cut off point.

No.	Sequence of removed and inserted edges	Change in distance for each pair of edges removed and added. Negative values are improvements. 1st	2nd	3rd	4th	5th
1	edges:109<-87+>98->107<+58->98+>74->100<+0->74+>109	-4565	1516	-49159	2870	12811
2	edges:109<-87+>98->107<+58->98+>74<-0+>100<-74+>109	-4565	1516	-49159	15681	0
3	edges:109<-87+>98<-58+>107<-98+>74->100<+0->74+>109	-4565	-32129	-15514	2870	12811
4	edges:109<-87+>98<-58+>107<-98+>74<-0+>100<-74+>109	-4565	-32129	-15514	15681	0
5	edges:74->100<+0->74+>109<-87+>98->107<+58->98+>74	2870	12811	-4565	1516	-49159
6	edges:74->100<+0->74+>109<-87+>98<-58+>107<-98+>74	2870	12811	-4565	-32129	-15514
7	edges:74<-0+>100<-74+>109<-87+>98->107<+58->98+>74	15681	0	-4565	1516	-49159
8	edges:74<-0+>100<-74+>109<-87+>98<-58+>107<-98+>74	15681	0	-4565	-32129	-15514
9	edges:87->109<+74->100<+0->74<+98->107<+58->98<+87	-11114	2870	15027	1516	-44826
10	edges:87->109<+74->100<+0->74<+98<-58+>107<-98<+87	-11114	2870	15027	-32129	-11181
11	edges:87->109<+74<-0+>100<-74<+98->107<+58->98<+87	-11114	15681	2216	1516	-44826
12	edges:87->109<+74<-0+>100<-74<+98<-58+>107<-98<+87	-11114	15681	2216	-32129	-11181
13	edges:98->107<+58->98<+87->109<+74->100<+0->74<+98	1516	-44826	-11114	2870	15027
14	edges:98->107<+58->98<+87->109<+74<-0+>100<-74<+98	1516	-44826	-11114	15681	2216
15	edges:98<-58+>107<-98<+87->109<+74->100<+0->74<+98	-32129	-11181	-11114	2870	15027
16	edges:98<-58+>107<-98<+87->109<+74<-0+>100<-74<+98	-32129	-11181	-11114	15681	2216

> and < indicate the direction travelled
- and + indicate if the edge was removed or added

Figure 6.2 - Many partial changes can locate the same improvement

For an example of how many partial changes can be used to locate a single improvement consider Figure 6.2. The table contains 16 ways of creating the same 5-opt improvement.

The figure 6.2 works as follows:

- A negative value means the combination of the removed and inserted edge resulted in an improvement. If we look at row No. 1, it begins by removing the edge between customer 109 and 87 and inserting an edge between customer 87 and 98. The –4565 tells us how much the distance travelled is reduced.
- The symbols > and < show which customer (node) the vehicle departed from and arrived at, i.e. the route direction.
- The symbols – and + show if the journey (edge) was added or removed.

The best results are achieved by move 15, because it allows the improvement to be identified using a high quality acceptance threshold. Move 15 begins by removing the edge that connects 98 to 58 and inserts an edge between 58 to 107, reducing distance travelled by -32129. The next edge pair removed and added reduce distance further giving a total reduction of -43310. With moves 15 and 16 the total reduction never falls below -32129. By using a high quality acceptance threshold of say -32000 for the total distance reduction, solutions 1-14 can be rejected as soon as the 1st pair of edges is tested. The advantage with this is that all the other solutions that start with these edge pairs are also rejected at the same time.

Improvements that could only be located by low quality changes turned out to be very small. As a typical rule of thumb the higher quality the change, the higher quality the improvement located by the change. This meant that a near optimal threshold was able to locate almost all the improvements in the neighbourhood. Even if the threshold fell a little way short of the optimal, most of the improvements missed would typically be small.

The improvement preference mechanism used in the main experiments gradually changes the threshold as the search progresses. Eleven threshold intervals were used in the main experiments. The formative experiments were used to assess how many thresholds to use and what value each should be set at. Because of the fact that a single improvement can be located using more than one partial change, a 10% threshold would only find approximately 10% of the improvements in the neighbourhood. It turned out that using 10%, 20% ... thresholds produced an even split of the available improvements. There was a little clustering but it did not follow

much of a pattern. Because of this, 10% intervals looked like they would work well with the main improvement preference and improvement location experiments. At the high and low end of the scale, 5% and 95% were used, see figure 6.1, giving 11 thresholds in all.

The data described here shows many different solutions can be used to help locate the same improvement. This means putting all the solutions from a neighbourhood into a spreadsheet is a difficult way of calculating thresholds, and is not recommended. The thresholds used in the main experiments were calculated in two stages: -

- First a hill climber was used and part way into the hill climb the neighbourhood was analysed to establish which neighbours where able to locate compound improvements, see section 5.4 for a detailed description and pseudo code. This produced the 11 thresholds.
- Second the preference mechanism was turned off and each improvement location mechanism was executed using each of the 11 thresholds, see graph 7.1 for typical results and see section 5.5 for the pseudo code. This allowed the identification of a good lower threshold.

6.4) The misleading compound move edge – Further work

The formative experiments highlighted one of the problems with using a small move to locate an improvement. Rather than create a move operator to locate 4-opt moves I used the 5-opt move operator. The 5-opt move was able to create 4-opt moves by adding and removing the same edge in a single move. This reduced the amount of coding needed. The observations here relate to partial changes rather than whole

moves, but the findings should apply to both. The problem was that the edge was inserted and then removed again. When this edge was used to locate improvements it made it hard to find several of the 4-opt improvements. As far as I was able to make out, using the edge was having a negative impact. I was able to locate all the improvements by using a high quality threshold if the extra edge was ignored. If the extra edge was used to locate improvements, a lower quality threshold was needed, causing the search to be less effective.

I did not code round the misleading edge problem because using a raised threshold was a simpler way round the problem. Coding round the problem with whole moves would also have made the main results less realistic as that is not how existing whole move methods work.

The misleading compound edge data suggests a possible avenue of further research. The problem is, when an edge (solution property) is used to help locate an improvement that it is not part of, it makes the improvement more difficult to locate. This suggests that the idea behind Lin-Kernighan [Lin 73] of using only solution properties that look like they will lead to an improvement does offer an advantage in terms of the trade off between algorithm speed solution quality. It looks like it is worth exploring the idea of creating a hybrid which uses whole move differences and partial changes. The difficulties with using slack time to help guide partial changes described in section 7.5.3 suggest a hybrid is worth looking into.

6.5) Using both distance and time window thresholds to guide the search

The formative experiments tested different ways of combining distance and time window thresholds to guide the search. The experiments in section 5.3 collected neighbourhood information allowing different ways of locating improvements to be evaluated.

The formative results indicated that the combined use of both distance and time thresholds to guide the search would be effective. Although the main hill climber results showed that using both thresholds did not offer a consistent advantage. The impact of combining the two is covered in more depth with main hill climber results in section 7.2 of the next chapter. The problem, I suspect, was in detecting if both thresholds were good at rejecting non-improving moves. The data that had been gathered about the neighbourhoods was good for deciding which thresholds would locate improvements. But the data could not be used to reliably predict the number of rejected non-improvements. This was because the 5-opt neighbourhood contained an excessively large number of non-improving solutions.

Regarding preference, the formative experiments did suggest that using a combination of the best distance threshold and a changing time window thresholds was a good way to control the preference. It was suspected this would be a good method because the analysis of the neighbourhood using the spreadsheet showed an even split in the number of improvements that each preference stage would be able to locate. When it came to the main experiments, the results were inconsistent. It was unclear if either of the two methods of combining the distance and slack time thresholds was better than the other, see sections 7.2 and 7.3.

The discussion above illustrates that putting all the solutions from a neighbourhood into a spreadsheet to calculate thresholds was difficult and unreliable. A better alternative was to first use a hill climber to calculate the percentile thresholds and then turn off the improvement preference mechanism. This was simpler and more accurate. The results discussed in 7.4.6 go further than this and suggest that the use of distance and slack time to guide the location and preference mechanisms should have been evaluated separately before they were combined.

6.6) Summary

The results described in this chapter support the idea of fine-tuning the improvement location acceptance threshold when the preference mechanism is turned off, this is described in sections 6.2, 6.3 & 6.5. The location mechanism's thresholds can be fine tuned while preference is turned on, although the results in sections 7.4.1 & 7.4.4 suggest that this is best done with the mechanism turned off. This also allows the impact of giving no preference to particular improvements be assessed.

7) Results of the VRP experiments

By distinguishing between the mechanisms of improvement location and preference, the results presented here show which combinations of location and preference do well and on which problems. Equally the results show which combinations work poorly together. Without this distinction it would be difficult to tell if a particular preference mechanism was having a negative impact on the improvement location mechanism being used. The reason the results can show which location and preference mechanisms work well together, is because the algorithms used distinguish between them. The results support the claim that distinguishing between location and preference is useful because doing so enabled the mechanisms that work well together to be identified.

Lin-Kernighan style partial moves were able to outperform the other methods when time windows were slack. They also offered an advantage when customers were clustered. Although, partial moves were difficult to implement and the percentage improvement was small, varying between 0.2% and 5%. Furthermore the results show that giving preference to small improvements in solution quality works well with all the VRPTW problem types tested. As with partial moves, the giving preference to small improvements in solution quality only produced slightly better final solutions. One good thing about switching to a small improvements first preference mechanism is, it is relatively easy modification.

Current local search algorithms for the VRPTW do not appear to locate improvements using Lin-Kernighan style partial moves nor do they appear to give preference to small improvements in solution quality. These unused location and

preference mechanisms produced the best quality final solutions on most of the VRPTW problem styles tested. The claim that the hypothesis is useful is supported by the fact that distinguishing between the mechanisms helped reveal the potential value of the mechanisms.

The first half of this chapter, see sections 7.1 –7.3, gives detailed descriptions of which methods, did well with which VRPTW problem types. The second half, sections 7.4 – 7.6, describe general findings, and suggests how existing meta-heuristics can be improved.

Execution time and the tightness of the Time Windows had the largest impact on algorithm performance. Based on this observation, the results have been divided into 6 groups, as shown in Table 7.1. The best methods were either very fast to start with and tailed off early or were slow to start with and produced the best quality final solutions, see graph 7.2 in section 7.4.7.

Fast/Slow	**Time window size**
Best early solutions: some methods were fast to begin with and tailed off early. These methods created good quality solutions earlier than the other slower methods.	**Large time window problems:** some methods typically did better when time windows were slack. Large time windows allow many customers per vehicle and so needed few vehicles.
Best final solutions: some methods tailed off later than others, thus creating higher quality final solutions. These methods were typically slower than the best, early solution methods.	**Small time window problems:** some methods typically did better when time windows were tight. Small time windows resulted in fewer customers per vehicles and so needed more vehicles.
Giving 6 groups	
1. Best early solutions, large time window problems – Table	
2. Best early solutions, small time window problems	
3. Point at which early solutions are overtaken, large time window problems	
4. Point at which early solutions are overtaken, small time window problems	
5. Best final solutions, large time window problems	
6. Best final solutions, small time window problems	

Table 7.1 – Key - 6 Result Groups

The experiments compared methods of locating improvements and deciding the improvement preference. The experiments compared both the solution quality and the speed of the methods.

The results showed that the following performed the best: -

- *High quality, partial distance changes first, produce the best early solutions with the large time window problems.*
- *No preference, whole distance moves, produce the best early solutions with the small time window problems.*
- *Low quality, partial distance changes first, produce the best final solutions with large time window problems.*
- *Low quality, partial distance changes first, produce the best final solutions with small time window problems.*

In some cases, the percentage difference between some of the averages was very small. This is especially true for the methods that produce the best quality final solutions.

7.1) Result Comparison Tables

The tables 7.4 to 7.13 show which local search methods performed best and on which problems, t-test results are also shown. The progress of each of the 20 local search methods is recorded over time. Tables 7.4 & 7.7 compare fast methods, tables 7.10 & 7.12 compare methods that produce the highest quality solutions and tables 7.5 & 7.7 show the point at which the slower methods, that produced higher quality

solutions, surpassed the fast methods. Each percentage is an average taken from 40 runs. The key: table 7.2, contains all the method variations used to locate improvements and how preference was given to particular improvements at the early stages of the search. The key: table 7.3, contains the 11 types of Vehicle Routing Problem with Time Windows (VRPTW) benchmarks that were used. Both the original Solomon benchmarks [Solomon 87] and the some of the extended Solomon benchmarks were used.

<table>
<tr><td colspan="6">Key - 20 local search guidance methods
See the 10 results tables below 7.4 – 7.13
The results tables only shows results for the best tail off thresholds
The methods use the difference in solution distance/time to work out the odds that a 3-opt move can be used to locate a 4 or 5-opt improvement.</td></tr>
<tr><td colspan="6">Preference methods</td></tr>
<tr><td colspan="6">N = No preference. Use all 3-opt improvements and some non-improving 3-opt solutions to find 5-opt improvements.
H = High – Initially use large 3-opt improvements to locate 5-opt improvements, gradually expanding to all.
L = Low – Initially use small 3-opt improvements and non-improving 3-opt solutions to locate 5-opt improvements, gradually expanding to all.</td></tr>
<tr><td colspan="6">Location methods – Number of changes used to create a solution estimate</td></tr>
<tr><td colspan="6">P = Part – Uses individual partial solution changes to locate 4 and 5-op improvements.
W = Whole – Uses 3-opt moves used to locate 4 and 5-opt improvements.</td></tr>
<tr><td colspan="6">Location methods – Information used to decide if the solution change should be used to try and locate an improvement.</td></tr>
<tr><td colspan="6">D = Distance – Use difference in solution distances to help locate improvements
T = Time – Use difference in solution slack time to help locate improvements
DT = Distance + Time – Uses linked distance and time thresholds. Where both threshold are a fixed number of percentiles below the optimal tail off point.
BDT = Best Distance + Time – Uses the best distance tail off threshold with different slack time thresholds.</td></tr>
<tr><td colspan="6">Used to create the 20 algorithms: -</td></tr>
<tr><td>NPD
NPT
NPDT
NPBDT</td><td>NWD
NWT
NWDT
NWBDT</td><td>HPD
HPT

HPBDT</td><td>HWD
HWT

HWBDT</td><td>LPD
LPT

LPBDT</td><td>LWD
LWT

LWBDT</td></tr>
</table>

Table 7.2 – Key - 20 local search guidance methods

One method is used from each box
e.g. **NPD** = **N**o preference **P**artial **D**istance

Key - 11 VRPTW Problem Types See the 10 results tables below 7.4 - 7.13		
C = Clustered **Rc** = Random Clustered **R** = Random	**1** = Small time windows **2** = Large Time Windows	**_1** = 100 Customers **_2** = 200 Customers

Table 7.3 – Key - 11 VRPTW Problem Types

One problem style is used from each box
e.g. **C1_2** = **C**lustered. **1** = small time windows, **_2**00 Customers

Two algorithm types were used to produce the results in tables 7.4 to 7.13. The no preference style, prefixed by N, relates to the algorithm described in section 5.5 "Finding improvements - Percentile tail off point". The High and Low quality first styles, prefixed by H and L respectively, relate to the algorithm described in section 5.6 "Guide improvement preference".

7.2) Detailed descriptions of the early VRPTW solutions

The focus in this section is on early, fast, solutions and how long it takes the slower methods to catch up and overtake. These fast methods tailed off early, with spreadsheet analysis showing they typically tailed off in under 0.5 seconds, the results are given in tables 7.4 & 7.7 below. The other slower methods that produced better quality solutions needed up to 150 seconds to catch up, see tables 7.5 & 7.8. These other methods that produced higher quality solutions are discussed in section 1.3.

7.2.1) High quality partial distance first did well with large time windows

The percentage of improvement was used in the result tables, 7.4 to 7.13, to simplify the comparison of the results. This is because the actual distances are large, and vary from problem type to problem type, making them difficult to compare. The result

tables are sorted by percentage of improvement, with the methods that produced the best percentage improvement at the top. To understand the tables, consider the left column of table 7.4, where C2_1 is the problem type, the C = Clustered customers, 2 = large time windows and _1 = 100 Customers. The number at the top of the column, 1952.6, is the length of the initial randomly constructed C2_1 solution that was used by all the various algorithms. The NPD is one of the 20 algorithms, where the letter N = No Preference, P = Partial change and D = Distance (the difference in solution distance). The letters indicate how the algorithm guided the improvement location mechanism. In this case No Preference mechanism was used, whereas with HPD, the next one down, both the preference and location mechanisms were guided by the difference in solution distance. The 0.5 seconds is the execution time, so the percentage improvement, e.g. the 57.9%, is the percentage improvement of the NPD method after 0.5 seconds. The 57.9% figure means that the NPD method improved the initial solution by 57.9%.

Early Solutions, Large Time Windows

Percentage improvement on initial solutions averaged over 40 runs

Time is in seconds

Initial Random Solution Distances									
1952.6		2841.5		9833.7		2652.6		10150.2	
C2_1	Time 0.5	Rc2_1	Time 0.5	Rc2_2	Time 0.5	R2_1	Time 0.5	R2_2	Time 0.5
NPD	57.9%	HPD	50.3%	HPD	33.9%	HPD	55.5%	HPD	47.8%
HPD	57.0%	NPD	47.5%	NPD	31.3%	NPD	50.8%	NPD	46.1%
NWDT	50.3%	NWDT	46.5%	NWD	27.2%	NWDT	48.8%	NWDT	40.9%
NWBDT	49.0%	NPDT	45.8%	NWDT	26.8%	NWD	46.8%	NWD	40.1%
NWD	47.7%	NWBDT	44.9%	NWBDT	25.8%	NWBDT	46.5%	NWBDT	39.7%
HWD	47.2%	NPBDT	44.0%	HWD	24.4%	HWD	45.6%	HWD	37.7%
NPDT	46.5%	NWD	42.7%	NWT	21.0%	NPBDT	42.1%	NWT	31.0%
NPBDT	44.5%	HWD	42.2%	NPDT	14.6%	NPDT	41.7%	NPDT	30.3%
NWT	40.5%	NWT	38.3%	NPBDT	12.3%	NWT	40.5%	NPBDT	26.8%
HWBDT	33.2%	HWBDT	26.4%	HWBDT	7.8%	HWBDT	24.1%	LWBDT	16.2%
LWBDT	26.7%	HWT	24.1%	HWT	7.5%	LWBDT	23.9%	HWBDT	15.1%
HWT	25.5%	LWBDT	22.9%	LWBDT	7.3%	HWT	20.8%	HWT	13.9%
LWT	20.7%	LPBDT	22.4%	LWT	5.3%	LWT	18.7%	LWT	10.8%
LPBDT	15.2%	LWT	17.9%	NPT	2.9%	LPBDT	17.1%	NPT	8.0%
NPT	12.3%	NPT	13.2%	LWD	1.9%	NPT	12.7%	LPBDT	4.7%
LWD	10.8%	LWD	8.5%	HPT	0.0%	LWD	6.0%	LWD	3.8%
LPD	0.8%	LPT	5.2%	HPBDT	0.0%	LPT	3.6%	LPT	1.8%
HPT	0.0%	HPBDT	4.2%	LPD	0.0%	LPD	3.5%	LPD	0.3%
HPBDT	0.0%	LPD	1.5%	LPT	0.0%	HPT	0.0%	HPT	0.0%
LPT	0.0%	HPT	0.0%	LPBDT	0.0%	HPBDT	0.0%	HPBDT	0.0%

Table 7.4 – Early Solutions, Large Time Windows

See tables 7.2 & 7.3 for key

The tables 7.4 and 7.7 show solution qualities for very short execution times. The solution qualities, shown in the tables, were collected just after the fastest methods tailed off. The fastest methods typically started to tail off in under 0.5 seconds, although they did continue to gradually improve. Table 7.4 shows results for large time windows. Large time windows allow more possible solutions, and enable shorter solutions to be created.

The actual solution distances for the results in tables 7.4 - 7.13 can be calculated by multiplying the initial solution distance by:

(100-percentage) / 100

Overview – early solutions with large time windows

Best: HPD

Good: NPD, NWDT, NWD

Bad: LPT, HPT, HPBDT, LPD

Both High quality Partial Distance and No preference Partial Distance (HPD and NPD) consistently performed the best. Giving preference to High quality Partial Distance, HPD, always performed the best or near to the best, when giving no preference, NPD, comes a close second to HPD. The reason partial changes performed well with large time windows appears to be because the partial change method is similar to Lin-Kernighan which performs well on the TSP, which is a similar style of problem. It is similar in that the VRPTW can be turned into a TSP by removing constraints.

Following closely behind were, whole distance (NWD) and whole distance + time (NWDT). In some cases giving preference to high quality performed well, but not giving a preference appears to be more consistent, see NWD and NWDT. With high quality first, some execution time was spent finding improvements that were discarded in the early stages. This possibly put the high quality first method at a slight disadvantage.

Using partial time as a guidance methods did not work, see NPT, HPT and LPT. Using whole moves guided by time did not perform too well either, always lagging at least 10% behind the best methods. Giving preference to low quality changes also did badly. It did not matter what guidance method was used when low quality changes where given preference, they all did badly in the early stages of the search.

Time taken for other methods to catch up

The methods that produce the best early solutions do well right from the start. They continue to dominate for the first few seconds of the run. Eventually the slower methods do catch up with the fast methods that tail off early. Table 7.5 shows solution qualities at the point when the slower methods catch up. It takes them about 10-30 seconds on the 100 customer problems and 1-2.5 minutes on the 200 customer problems. These slower methods move from the bottom of the table to the top, compare table 7.4 with 7.10 and 7.7 with 7.12. That is they go from being the worst performing methods to the best performing in a matter of minutes, or seconds with the smaller problem.

Point at which early solutions are overtaken, Large Time Windows
Percentage improvement on initial solutions averaged over 40 runs
Time is in seconds

	Initial Random Solution Distances								
	1952.6		2841.5		9833.7		2652.6		10150.2	
	C2_1	Time 30	Rc2_1	Time 10	Rc2_2	Time 150	R2_1	Time 11	R2_2	Time 50
NPDT	64.1%	LPBDT	57.8%	HWT	70.1%	HPD	61.0%	HWBDT	68.0%
HPD	63.9%	NPD	57.6%	HWBDT	70.0%	NPD	60.7%	NPD	67.9%
NPD	63.9%	HPD	57.4%	LPBDT	69.9%	HWBDT	60.7%	HPD	67.7%
LPBDT	63.7%	HWBDT	57.4%	NPD	69.8%	NWD	60.6%	LWBDT	67.6%
HPBDT	63.4%	NPBDT	57.3%	HPD	69.7%	NWDT	60.3%	HWT	67.3%
NPBDT	63.3%	NWDT	57.2%	NPDT	69.6%	HWD	60.1%	NWDT	67.3%
LWBDT	62.7%	NPDT	57.1%	HPBDT	69.6%	NWBDT	60.1%	HWD	67.2%
HWBDT	62.7%	NWT	57.1%	NWDT	69.5%	LWBDT	60.1%	NWD	67.2%
LPD	62.6%	NWBDT	57.0%	NPBDT	69.3%	NWT	59.8%	NWT	67.1%
HWD	62.4%	LWBDT	56.9%	LWBDT	69.1%	HWT	59.8%	NWBDT	66.9%
HWT	62.4%	HWD	56.8%	HWD	69.1%	LPD	59.2%	LWD	66.3%
LWT	62.4%	NWD	56.8%	NWD	68.8%	LWT	58.6%	LWT	65.9%
LWD	62.2%	HWT	56.5%	NWBDT	68.8%	LPBDT	58.2%	LPBDT	62.9%
NWD	62.1%	LPD	55.4%	NWT	68.6%	NPDT	56.9%	NPBDT	61.4%
NWBDT	62.1%	LWD	55.2%	LWD	68.6%	NPBDT	56.7%	NPDT	61.3%
NWDT	62.1%	LWT	55.1%	LWT	67.8%	HPBDT	55.0%	HPBDT	58.8%
NWT	62.0%	HPBDT	53.3%	NPT	66.3%	LWD	54.1%	NPT	49.4%
NPT	60.2%	NPT	50.1%	LPT	42.4%	NPT	50.6%	LPD	45.7%
LPT	43.4%	LPT	32.8%	HPT	28.5%	LPT	34.2%	LPT	34.7%
HPT	32.8%	HPT	17.4%	LPD	19.6%	HPT	24.5%	HPT	26.2%

Table 7.5 – Point at which early solutions are overtaken, Large Time Windows

See tables 7.2 & 7.3 for key

Tables 7.5 and 7.8 show the typical execution time needed for the best performing methods to over take the early front runners. The time taken is mainly dependent on problem size, 100 customer columns end _1 and 200 customer problems end _2. The reasons for the difference appear to be mainly down to preference and are discussed in section 7.4. There is also some variation from problem style to problem style.

There is a consistent difference in performance with the early solution results. The p-values indicate how similar the methods are, the closer the p-value is to one the more similar the results of the methods. The statistical difference between the early

solutions is typically high, see tables 7.6 & 7.9. This means the methods that did best should consistently outperform the others. The higher the t-value the more consistent the difference.

The standard deviations for the results are in the appendix. If desired these can be used to calculate further p-values.

t-test: Early Solutions, Large Time Windows

Statistical Difference between the best performing method and the next four

C2_1	Time	Rc2_1	Time	Rc2_2	Time	R2_1	Time	R2_2	Time
	0.5		0.5		0.5		0.5		0.5
NPD	57.9%	HPD	50.3%	HPD	33.9%	HPD	55.5%	HPD	47.8%
p-values		p-values		p-values		p-values		p-values	
HPD	0.63	NPD	0.28	NPD	0.16	NPD	0.07	NPD	0.65
NWDT	0.00	NWDT	0.08	NWD	0.00	NWDT	0.00	NWDT	0.01
NWBDT	0.00	NPDT	0.17	NWDT	0.00	NWD	0.00	NWD	0.00
NWD	0.00	NWBDT	0.01	NWBDT	0.00	NWBDT	0.00	NWBDT	0.00

Table 7.6 – t-test: Early Solutions, Large Time Windows

See tables 7.2 & 7.3 for key

With bigger VRPTW problems, with over 200 customers, the expected execution time can become excessive. The results in this section show which methods can be used in order to help minimise excessive execution time on VRPTW problems that have large time windows and large numbers of customers. The results show that methods using the Partial Distance improvement location mechanism were best able to minimise execution times but only when combined with either the High quality or No preference mechanisms. These results support the claim that the hypothesis is useful because distinguishing between the methods allowed the results to show which mechanism combinations were best able to minimise execution times with the large time window VRPTW problems.

7.2.2) No preference whole distance moves did well with small time windows

Overview – early solutions with small time windows

Best: NWDT

Good: NWBDT, NWT, NWD

Bad: LPT, HPT, LPD, HPBDT

When the execution time was at 0.5 seconds, one whole move method dominated all the small time window problems. The dominant no preference method was guided by both distance and slack time, NWDT, and was faster than the other whole solution and partial solution methods. After several seconds the slower methods were able to catch up and over take this faster method.

Even though NWDT dominated, NWBDT, which is very similar, consistently came a close second. Both methods use a combination of distance and slack time to locate improvements. With NWDT, both thresholds are a fixed number of percentiles below the optimal tail off point. NWBDT uses the best distance tail off threshold with many different slack time thresholds. The one that produced the best quality solutions is then used in the tables.

With small time window VRP style problems where solutions need to be created in a few seconds, the results suggest that whole moves are better. It appears finding improvements using whole moves that use all the constraints to find improvements is a good way of finding early small time window improvements. This method tailed after a few seconds allowing some of the slower methods to catch up and overtake. With the larger 200 customer problems, this took between 30-60 seconds. With the

smaller 100 customer problems, it took between 4-12 seconds for the slow methods to catch up and over take.

All the methods that used no preference whole moves (N, W) consistently performed well. Using whole moves guided by distance and time was the fastest, with best whole distance time coming a close second. These fast performance results were achieved by not giving a preference to high or low quality solutions.

Whole moves that gave preference to high or low quality time did badly. When no preference whole time is compared to the two preference methods, both of the preference methods are outperformed by the no preference method in every case with early solutions. The no preference method, NWT, stands out because it consistently outperforms both preference styles rather than because it stands out from the rest. This could be because the no preference method does not reject improvements. But if rejection of improvements was the only factor then I would expect all of the no preference methods to dominate, and yet the reverse is true. It appears that no preference is better when using slack time to guide the search with small time windows when a very short amount of execution time is available.

Giving preference to low quality changes caused all methods to perform slowly on small time window problems.

All the methods guided by partial changes were slow and could not compete in the early stages of the search. The best two were both guided by distance, one used no

preference and the other gave preference to high quality changes first. Using partial time to guide the search also did badly.

Early Solutions, Small Time Windows

Percentage improvement on initial solutions averaged over 40 runs

Time is in seconds

				Initial Random Solution Distances							
2152.9		13000.5		2697.4		9729.7		2780.7		10467.0	
C1_1	Time 0.5	C1_2	Time 0.5	Rc1_1	Time 0.5	Rc1_2	Time 0.5	R1_1	Time 0.5	R1_2	Time 0.5
NWDT	44.9%	NWDT	54.3%	NWDT	41.2%	NWDT	35.1%	NWDT	46.6%	NWDT	32.8%
NWBDT	41.9%	NWBDT	51.4%	NWBDT	40.3%	NWBDT	31.9%	NWBDT	45.5%	NWBDT	31.0%
NWT	33.5%	NWT	40.9%	NWT	35.7%	NWD	26.4%	NWT	40.0%	NWD	24.4%
HPD	33.0%	NWD	38.6%	NPD	31.4%	NWT	26.2%	NWD	35.9%	NWT	24.1%
NWD	32.2%	NPD	33.4%	HWBDT	31.0%	HPD	25.2%	NPD	34.3%	NPD	22.8%
NPD	31.8%	HWD	29.8%	NWD	30.9%	NPD	24.7%	HWBDT	34.1%	HPD	21.9%
HWD	30.4%	HPD	28.3%	HWT	28.3%	NPBDT	19.7%	NPBDT	30.4%	HWD	21.2%
HWBDT	29.9%	LWBDT	25.4%	NPDT	28.2%	NPDT	19.2%	HWT	30.1%	NPDT	17.0%
NPDT	27.0%	HWBDT	23.4%	NPBDT	27.2%	HWBDT	14.7%	NPDT	29.5%	HWBDT	13.1%
NPBDT	24.9%	HWT	20.8%	LWBDT	23.5%	HWD	14.1%	HWD	28.9%	HWT	12.3%
HWT	23.6%	NPBDT	19.4%	HPD	21.4%	HWT	13.5%	HPD	28.5%	NPBDT	10.9%
LWBDT	23.3%	NPDT	18.7%	HWD	20.7%	LWBDT	7.5%	LWBDT	26.1%	LWBDT	8.2%
LWT	17.7%	LWT	17.4%	LWT	20.0%	LWT	7.1%	LWT	22.1%	LWT	7.5%
LPBDT	16.0%	NPT	8.5%	LWD	12.5%	NPT	4.4%	NPT	12.3%	NPT	4.9%
LWD	12.7%	LWD	7.6%	NPT	10.3%	LWD	2.6%	LWD	11.2%	LWD	3.1%
NPT	9.5%	LPBDT	1.2%	LPBDT	10.0%	HPT	2.0%	HPBDT	11.1%	LPBDT	0.3%
LPT	2.2%	HPT	0.0%	HPBDT	6.2%	LPD	0.6%	LPBDT	5.0%	HPT	0.0%
LPD	0.1%	HPBDT	0.0%	LPT	1.2%	HPBDT	0.0%	LPD	2.0%	HPBDT	0.0%
HPT	0.0%	LPD	0.0%	LPD	1.1%	LPT	0.0%	LPT	1.3%	LPD	0.0%
HPBDT	0.0%	LPT	0.0%	HPT	0.0%	LPBDT	0.0%	HPT	0.0%	LPT	0.0%

Table 7.7 – Early Solutions, Small Time Windows

See tables 7.2 & 7.3 for key

Table 7.7 shows results for small time windows. The methods with a letter, T, make use of the amount of slack time between customers with the aim of improving their ability to deal with time window constraints. Because of this these methods are of particular interest when time windows are small. Section 7.4.5 shows how giving preference to small differences in solution slack time improved final solution quality with the larger 200 customer problems.

Time taken for other methods to catch up

As with large time window problem, the methods that produce the best early solutions do well right from the start. Again, eventually the slower methods to catch up, see table 7.8. The difference with the small time windows is that the slower methods require much less time to catch up and overtake. It takes them about 4-12 seconds on the 100 customer problems and 38-50 seconds on the 200 customer problems.

Point at which early solutions are overtaken, Small Time Windows
Percentage improvement on initial solutions averaged over 40 runs
Time is in seconds

				Initial Random Solution Distances							
2152.9		13000.5		2697.4		9729.7		2780.7		10467.0	
C1_1	Time 4	C1_2	Time 50	Rc1_1	Time 12	Rc1_2	Time 38	R1_1	Time 9	R1_2	Time 40
NWBDT	55.6%	NPBDT	75.2%	HPD	46.5%	HWBDT	65.5%	HWBDT	52.3%	HPD	62.8%
NWDT	55.3%	HWBDT	75.2%	HWT	46.2%	NPDT	64.9%	NPD	52.2%	HWBDT	62.6%
NPD	55.0%	HWT	75.1%	LWBDT	46.1%	NPD	64.9%	HPD	52.2%	NPD	62.2%
NPBDT	55.0%	NPD	75.1%	NPD	46.1%	NWDT	64.7%	NWDT	52.1%	NWDT	62.0%
HWBDT	54.6%	LPBDT	75.0%	HWBDT	46.0%	LPD	64.7%	HWT	52.1%	LWBDT	61.9%
NPDT	54.0%	LWBDT	75.0%	NWT	45.8%	HWT	64.6%	HWD	52.0%	NWD	61.7%
HPD	53.9%	HPD	75.0%	NWD	45.7%	LPBDT	64.5%	NWD	51.9%	NWT	61.6%
NWD	53.6%	NPDT	75.0%	HWD	45.7%	HPD	64.4%	LWBDT	51.9%	HWD	61.5%
NWT	53.4%	NWBDT	74.8%	LWD	45.6%	NWBDT	64.4%	LPD	51.7%	NWBDT	61.4%
HWD	51.6%	HWD	74.8%	LWT	45.6%	NWT	64.0%	NWT	51.7%	HWT	61.4%
LWBDT	51.3%	NWD	74.7%	NWBDT	45.5%	NWD	63.9%	LWD	51.4%	LWT	60.2%
LPBDT	50.3%	NWT	74.7%	NWDT	45.4%	NPBDT	63.9%	NWBDT	51.4%	LWD	58.9%
HWT	49.8%	NWDT	74.6%	LPD	45.2%	HWD	63.4%	LWT	51.3%	NPDT	50.6%
LWT	45.0%	LWD	74.2%	LPBDT	45.0%	LWBDT	62.2%	LPBDT	48.0%	NPBDT	50.5%
LWD	40.8%	LWT	73.3%	NPDT	44.8%	LWT	61.3%	NPDT	47.9%	LPBDT	43.5%
NPT	32.9%	NPT	70.0%	NPBDT	44.7%	LWD	59.4%	NPBDT	47.8%	NPT	38.6%
LPT	16.0%	LPT	33.6%	HPBDT	41.5%	NPT	55.8%	NPT	44.0%	HPBDT	32.7%
HPBDT	13.1%	HPBDT	30.5%	NPT	40.9%	HPBDT	39.5%	HPBDT	35.5%	HPT	21.6%
LPD	9.4%	HPT	18.9%	HPT	22.3%	HPT	19.3%	HPT	22.5%	LPT	15.9%
HPT	7.3%	LPD	18.5%	LPT	18.5%	LPT	13.4%	LPT	18.5%	LPD	9.5%

Table 7.8 – Point at which early solutions are overtaken, Small Time Windows

See tables 7.2 & 7.3 for key

As pointed out with the large time window solutions, there is a consistent difference in performance with the early solution results. Because of this, the best of these methods can be expected to consistently outperform the others tested, this is shown in table 7.9.

t-test: Early Solutions, Small Time Windows

Statistical Difference between the best performing method and the next four

C1_1	Time	C1_2	Time	Rc1_1	Time	Rc1_2	Time	R1_1	Time	R1_2	Time
	0.5		0.5		0.5		0.5		0.5		0.5
NWDT	44.9%	NWDT	54.3%	NWDT	41.2%	NWDT	35.1%	NWDT	46.6%	NWDT	32.8%
p-values		p-values		p-values		p-values		p-values		p-values	
NWBDT	0.02	NWBDT	0.00	NWBDT	0.60	NWBDT	0.00	NWBDT	0.58	NWBDT	0.09
NWT	0.00	NWT	0.00	NWT	0.00	NWD	0.00	NWT	0.00	NWD	0.00
HPD	0.00	NWD	0.00	NPD	0.00	NWT	0.00	NWD	0.00	NWT	0.00
NWD	0.00	NPD	0.00	HWBDT	0.00	HPD	0.00	NPD	0.00	NPD	0.00

Table 7.9 – t-test: Early Solutions, Small Time Windows

See tables 7.2 & 7.3 for key

Again problems with a large number of customers can have excessive execution times. The results in this section are for the VRPTW problems that have small time windows. They show which methods can be used in order to help minimise excessive execution time. The results show that methods using the Whole Distance and Time as an improvement location mechanism were best able to minimise execution times but only dominated when combined with no preference. Again these results support the claim that the hypothesis is useful. This is because distinguishing between the methods allowed the results to show which mechanism combinations were best able to minimise execution times with the small time window VRPTW problems.

7.3) Detailed descriptions of the best quality VRPTW solutions

If an execution time of several minutes is available then the results suggest that the methods highlighted here will perform well. The slowest of the 200 customer methods had all tailed off and stopped improving after 22 minutes. The 100 customer methods had all finished in under 1 minute.

With some problems very fast solutions are needed, but with other problems an execution time of several minutes or even several hours is acceptable. This section looks at the methods with a high solution quality tail off point. Most of the early solution methods tail off when solution quality is quite low. This section discusses the methods that produced the best solution qualities when given more execution time.

7.3.1) Low quality partial distance first did well with large time windows

Overview – best solutions with large time windows

Best LPD

Good LWD, NPD, HPD

Bad: LPT, HPT

No preference Partial Distance (NPD) and High quality first Partial Distance (HPD) consistently performed well. These two methods also performed well in the early stages of the search as well as having a high quality tail off. Low quality Partial Distance (LPD) did not follow this pattern, it went from being one of the worst performing methods to being the best, see early and best result tables 7.4, 7.7, 7.10 & 7.12.

Low quality first whole moves guided by distance (LWD) also performs well. Like LPD it starts of slow and eventually overtakes the faster methods that tail off early. Although for some unknown reason LWD performed badly on one problem style, large time windows with 100 clustered customers.

In the early stages of the search NWDT was one of the best performing methods but tailed off early and was over taken.

Methods guided by partial time, LPT, NPT and HPT consistently came last or near to the last. They performed badly both at the start of the search and when they tailed off. Out of the three methods, NPT consistently outperformed the other two.

Best quality solutions, Large Time Windows

Percentage improvement on initial solutions averaged over 40 runs

Time is in seconds

Initial Random Solution Distances									
1952.6		2841.5		9833.7		2652.6		10150.2	
C2_1	Time 50	Rc2_1	Time 50	Rc2_2	Time 1300	R2_1	Time 50	R2_2	Time 164
NPDT	64.1%	LPD	58.9%	LPD	70.4%	LPD	62.0%	LWD	68.5%
LPD	64.0%	LWD	58.0%	HWT	70.2%	LWD	61.9%	LPD	68.1%
HPD	63.9%	LPBDT	57.8%	LWD	70.1%	HPD	61.0%	HWBDT	68.0%
NPD	63.9%	NPD	57.6%	HWBDT	70.0%	HWT	60.8%	NPD	67.9%
LPBDT	63.7%	LWBDT	57.6%	LPBDT	69.9%	NPD	60.7%	HPD	67.7%
HPBDT	63.4%	HPD	57.4%	HPBDT	69.8%	HWBDT	60.7%	LWBDT	67.6%
NPBDT	63.3%	HWBDT	57.4%	NPD	69.8%	NWD	60.6%	HWT	67.6%
LWBDT	62.7%	HPBDT	57.3%	HPD	69.7%	LWT	60.5%	NWT	67.3%
HWBDT	62.7%	NPBDT	57.3%	NPDT	69.6%	LWBDT	60.4%	NWDT	67.3%
LWT	62.6%	NWT	57.3%	NWDT	69.5%	NWDT	60.3%	LWT	67.2%
HWD	62.4%	NWDT	57.2%	NPBDT	69.3%	HWD	60.2%	HWD	67.2%
HWT	62.4%	NPDT	57.1%	LWT	69.3%	NWT	60.1%	NWD	67.2%
LWD	62.2%	HWT	57.1%	LWBDT	69.2%	NWBDT	60.1%	NWBDT	66.9%
NWD	62.1%	HWD	57.0%	HWD	69.1%	LPBDT	58.4%	LPBDT	62.9%
NWBDT	62.1%	LWT	57.0%	NPT	68.9%	HPBDT	57.5%	HPBDT	61.8%
NWDT	62.1%	NWBDT	57.0%	NWD	68.8%	NPDT	57.0%	NPDT	61.6%
NWT	62.0%	NWD	56.8%	NWBDT	68.8%	NPBDT	56.7%	NPBDT	61.4%
NPT	61.3%	NPT	56.1%	NWT	68.7%	NPT	56.0%	NPT	56.3%
LPT	46.4%	HPT	44.1%	HPT	66.6%	LPT	47.7%	LPT	49.0%
HPT	38.9%	LPT	42.8%	LPT	63.3%	HPT	41.0%	HPT	36.8%

Table 7.10 – Best quality solutions, Large Time Windows

See tables 7.2 & 7.3 for key

Tables 7.10 and 7.12 show the best quality results. These were collected just after the methods tailed off. The tail off point was detected by an excessive number of consecutive failures to find an improvement.

Some methods produced solutions slightly better than those shown in tables 7.10 and 7.12. The worst performing methods often had both excessive execution times and low quality solutions. These unsuccessful methods had slightly better results than the ones shown because the solutions were collected before these methods tailed off.

The 5 methods that create the best quality solutions look to be almost the same, see tables 7.11 & 7.13. In most cases the best performing method cannot be expected to consistently out perform the other 5. The p-values indicate that there is little to pick between top few of the best performing methods and several execution runs would be needed to identify a difference in performance.

t-test: Best quality solutions, Large Time Windows

Statistical Difference between the best performing method and the next four

C2_1	Time	Rc2_1	Time	Rc2_2	Time	R2_1	Time	R2_2	Time
	50		50		1300		50		164
NPDT	64.1%	LPD	58.9%	LPD	70.4%	LPD	62.0%	LWD	68.5%
	p-values		p-values		p-values		p-values		p-values
LPD	0.77	LWD	0.57	HWT	0.75	LWD	0.96	LPD	0.78
HPD	0.74	LPBDT	0.51	LWD	0.61	HPD	0.45	HWBDT	0.74
NPD	0.69	NPD	0.42	HWBDT	0.53	HWT	0.35	NPD	0.67
LPBDT	0.50	LWBDT	0.40	LPBDT	0.42	NPD	0.35	HPD	0.58

Table 7.11 – t-test: Best quality solutions, Large Time Windows

See tables 7.2 & 7.3 for key

Results for large time windows are covered in this section with the focus placed on methods that produce the best quality final solutions. Using distance to guide both the improvement location mechanism and the preference mechanism dominated these results. Using partial changes to locate improvements was able to produce best or near best results using any of the three preference mechanisms; although giving preference to low quality changes worked the best with partial changes. If a whole move improvement location mechanism is used, because they are simpler to implement than partial changes, then combining it with the low quality preference mechanism enabled the method to produce results that were among the best. Distinguishing between the mechanisms allowed the results to show which

mechanism combinations were best able to produce the best quality final results with the large time window VRPTW problems.

7.3.2) Low quality partial distance first did well with small time windows

Overview – best solutions with small time windows

Best LPD, LWD

Good NPD

Bad LPT, HPT

With the small time window results no one method dominates the best quality results. There is little to choose between Low quality Partial Distance (LPD) and Low quality Whole Distance (LWD). LPD beats LWD in 4 of the 6 problems.

Following closely behind these two was no preference partial distance NPD. NPD also perform inconsistently, doing well on some problems as not so well on others. It appears to perform better on the smaller 100 customer problems than the larger 200 customer problems.

Partial solutions were able to outperform whole on both the 100 clustered customer problems, C1_1 and C2_1. Both LWD and LPD do not perform well on the small time window version of the 100 clustered customer problem, C1_1. NPD produced better quality final solutions than all the whole move methods with both large and small time windows. Also, unlike LWD, LPD performs well with the large time window version, C2_1.

Methods guided by partial time followed the same pattern as large time windows, with LPT, NPT and HPT consistently coming last or near to the last. Again, out of the three methods, NPT consistently outperformed the other two. Section 7.5.3 suggests that the poor performance of the partial time methods was because the estimate of slack time was rough and needed to be more accurate.

Best quality solutions, Small Time Windows
Percentage improvement on initial solutions averaged over 40 runs
Time is in seconds

				Initial Random Solution Distances							
2152.9		13000.5		2697.4		9729.7		2780.7		10467.0	
C1_1	Time 50	C1_2	Time 420	Rc1_1	Time 50	Rc1_2	Time 240	R1_1	Time 25	R1_2	Time 600
NPD	57.9%	LPD	76.1%	HPD	46.5%	LPD	66.2%	LPD	53.1%	LWD	63.3%
LPBDT	57.8%	LWD	76.0%	LWD	46.5%	HWBDT	65.6%	LWD	52.7%	HPD	62.9%
NPBDT	57.4%	LPBDT	75.9%	NPD	46.3%	HWT	65.4%	HPD	52.5%	LPD	62.9%
HPD	57.2%	HWT	75.3%	HWT	46.2%	LWD	65.4%	NPD	52.4%	HWBDT	62.6%
NPDT	57.0%	NPBDT	75.2%	LWBDT	46.1%	LPBDT	65.2%	HWBDT	52.3%	NPD	62.5%
LPD	57.0%	HWBDT	75.2%	HWBDT	46.0%	NPD	65.1%	HWD	52.1%	HWT	62.5%
LWD	56.6%	NPD	75.2%	LPD	45.9%	NPDT	64.9%	NWDT	52.1%	LWBDT	62.5%
LWBDT	56.6%	LWBDT	75.1%	NWT	45.8%	HPBDT	64.8%	LWT	52.1%	NWT	62.1%
HPBDT	56.4%	HPD	75.0%	NWD	45.7%	NWDT	64.7%	HWT	52.1%	LWT	62.1%
LWT	56.4%	NPDT	75.0%	HWD	45.7%	LWT	64.6%	NWD	52.0%	NWDT	62.0%
NWT	55.7%	HPBDT	75.0%	LWT	45.6%	NWD	64.5%	LWBDT	51.9%	NWD	61.9%
NWBDT	55.6%	NWBDT	74.8%	NWBDT	45.5%	LWBDT	64.5%	NWT	51.8%	HWD	61.8%
HWT	55.6%	NWT	74.8%	NWDT	45.4%	HPD	64.4%	NWBDT	51.4%	NWBDT	61.4%
NWDT	55.6%	LWT	74.8%	LPBDT	45.0%	HWD	64.4%	LPBDT	48.5%	LPBDT	58.5%
NWD	55.5%	HWD	74.8%	HPBDT	45.0%	NWBDT	64.4%	NPDT	48.4%	NPDT	57.5%
HWBDT	55.4%	NWD	74.7%	NPDT	44.8%	NWT	64.3%	NPBDT	48.1%	HPBDT	57.4%
HWD	55.4%	NWDT	74.6%	NPBDT	44.8%	NPBDT	63.9%	HPBDT	47.5%	NPBDT	56.7%
NPT	55.3%	NPT	74.2%	NPT	43.9%	NPT	62.9%	NPT	46.3%	NPT	52.6%
LPT	35.2%	LPT	62.2%	LPT	38.7%	HPT	34.2%	LPT	31.3%	LPT	45.2%
HPT	28.7%	HPT	49.6%	HPT	38.4%	LPT	33.2%	HPT	30.1%	HPT	41.3%

Table 7.12 – Best quality solutions, Small Time Windows

See tables 7.2 & 7.3 for key

As pointed out with the best quality large time window solutions, the best performing method can not be expected to consistently outperform the others and several execution runs would be needed to identify a difference in performance.

t-test: Best quality solutions, Small Time Windows

Statistical Difference between the best performing method and the next four

C1_1	Time	C1_2	Time	Rc1_1	Time	Rc1_2	Time	R1_1	Time	R1_2	Time
	50		420		50		240		25		600
NPD	57.9%	LPD	76.1%	HPD	46.5%	LPD	66.2%	LPD	53.1%	LWD	63.3%
p-values		p-values		p-values		p-values		p-values		p-values	
LPBDT	0.96	LWD	0.65	LWD	0.99	HWBDT	0.50	LWD	0.81	HPD	0.85
NPBDT	0.58	LPBDT	0.40	NPD	0.91	HWT	0.37	HPD	0.72	LPD	0.82
HPD	0.39	HWT	0.03	HWT	0.86	LWD	0.36	NPD	0.70	HWBDT	0.72
NPDT	0.35	NPBDT	0.00	LWBDT	0.83	LPBDT	0.26	HWBDT	0.64	NPD	0.66

Table 7.13 – t-test: Best quality solutions, Small Time Windows

See tables 7.2 & 7.3 for key

Results for small time windows are covered in this section, with the focus placed on methods that create the best quality final solutions. The mechanisms that produced the best quality solutions with large time windows also tended to produce the best quality solutions with small time windows. Again the low quality first preference mechanism typically produced the best quality solutions and both of the location mechanisms that were guided by distance tended to dominate. Distinguishing between the mechanisms allowed the results to show which mechanism combinations were best able to produce the best quality final results with the small time window VRPTW problems.

7.4) Improvement preference – general findings

The choice of improvement location and improvement preference mechanism is dependent on the problem being solved and the how much time can be spent creating a solution. The results in this section and the following sub sections show that preference is having an impact on solution quality and execution time. This

supports the claim that is it useful to distinguish between the two mechanisms because it shows that the choice of preference mechanism makes a difference to solution quality and execution time.

The preference experiments were designed to detect improvements in solution quality rather than speed. Because of this the speed of the methods that gave a preference is probably below optimal. Fine-tuning could be done that would potentially improve speed. The preference experiments gradually change the acceptance threshold. The rate at which this threshold is changed could be looked at and potentially improved. It is unclear how much would be gained by doing this.

It was expected that all the methods that gave preference to one improvement rather than another would be slower than the methods that gave no preference because they would spend time rejecting improvements that could be implemented. As noted above in section 7.2.1, giving preference to high quality first was faster with most of the large time window problems tested.

Better quality solutions were achieved by giving preference to some improvements rather than others. The experiments tested giving preference to low quality moves and high quality moves to see what impact this had. The experiments start off by using only low or high quality moves and gradually increasing to all. This offered a gain of between 1.05% and 3.26% over using no preference for 100 customer problems. For 200 customer problems gain of between 2.4% and 4.5% was achieved. See table 7.14 for details of what was gained by giving preference to high or low quality changes. These percentages are based on comparing the best solution

quality no preference method with the best preference method. The details of which preference methods produced better quality final results see tables 7.10 & 7.12 above.

When a preference was given to high or low quality changes							
	Clustered with Large Time Windows	Clustered with Small Time Windows	Random & Clustered with Large Time Windows	Random & Clustered with Small Time Windows	Random Distribution with Large Time Windows	Random Distribution with Small Time Windows	Average
100 Customer Problems – How much was gained by giving a preference to high or low quality changes							
Whole Moves	1.36%	1.99%	1.49%	1.23%	3.26%	1.05%	1.73%
Partial Change	-0.43%	0.00%	2.91%	0.55%	3.36%	1.66%	1.34%
Both whole and partial	4.88%	4.93%	3.71%	1.30%	3.45%	2.03%	3.38%
200 Customer Problems - How much was gained by giving a preference to high or low quality changes							
Whole Moves		4.50%	2.40%	2.47%	3.53%	3.03%	3.18%
Partial Change		0.83%	2.15%	3.15%	0.67%	1.17%	1.60%
Both whole and partial		5.08%	3.20%	4.28%	3.53%	3.03%	3.82%

Table 7.14 - Percentage gain in solution distance with sequence experiments

7.4.1) Damaging effect of very low quality improvements

Very low quality changes had a negative effect on quality. This occurred when the methods use low quality previous solutions to locate improvements. When the acceptance threshold is low the solution quality gets worse. Look at the R2 location results, graph 7.1, and look at how 0_whole (also see "description of key" below) does worse than 1_whole and 0_Partial does worse than 1_Partial. Both of the methods that do worse, allow more of the available improvements to be

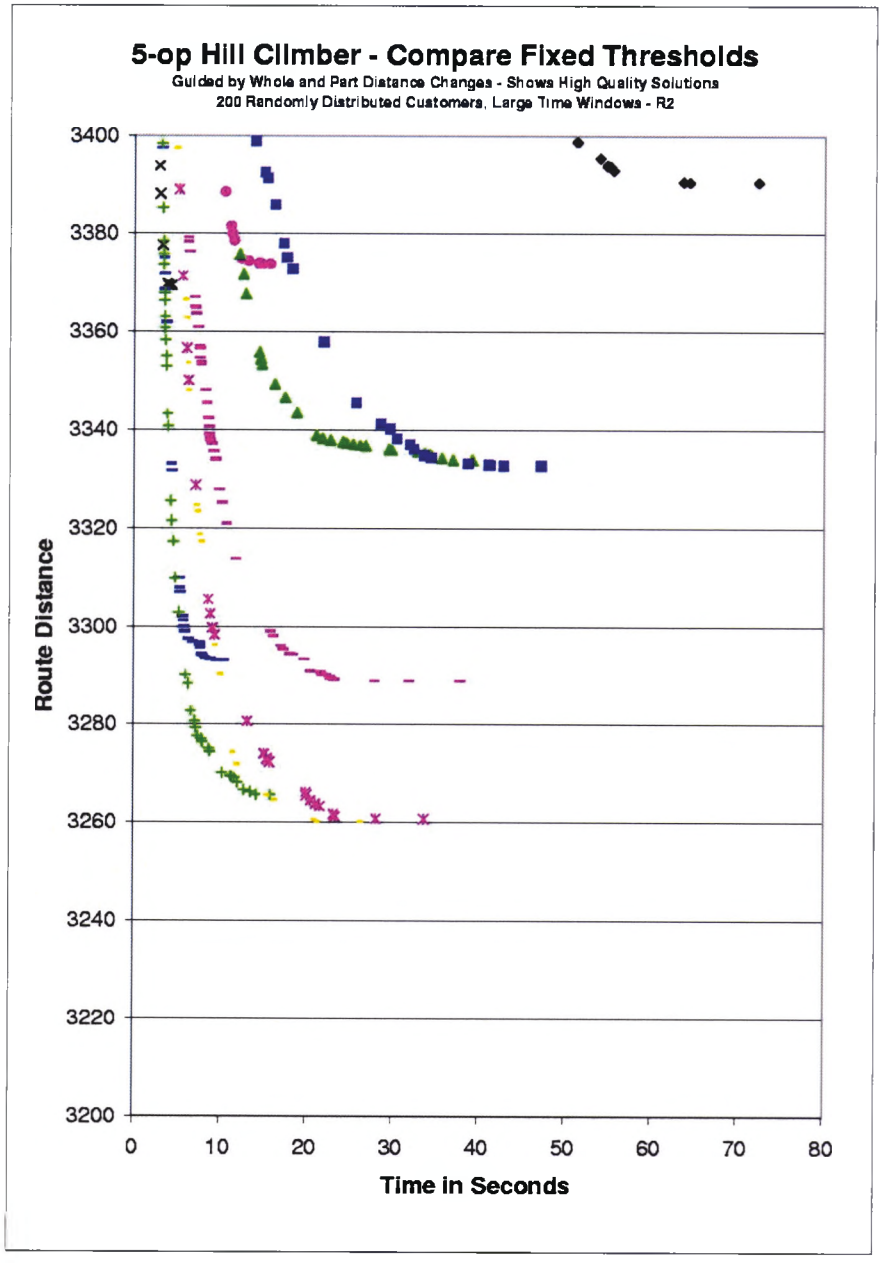
implemented. It looks as if allowing lower quality previous solutions to be used damages solution final quality.

Graph 7.1 (below) description of key

- The leading 0_ .. 10_ indicate the threshold cut off point used. Where _0 means the method is expected to locate at least 5% of the available improvements in the neighbourhood, although the 5% that are the easiest to find. The percentages used are 5% 10% 20% 30% ...80% 90% 95% these relate to the 0_ 1_ 2_ 3_ ... 8_ 9_ 10_. A more detailed description of percentile tail off can be found in section 5.5 "Finding improvements – Percentile tail off point".

- "Whole" indicates that the method used whole previous solutions to locate improvements and "Partial" indicates that partial solution changes were used to locate improvements.

- "Vehicles=" indicates the average number of vehicles per solution.



Graph 7.1 – Location results – damaging effect of low quality improvements

Lowering the acceptance threshold causes better solutions to be found, but this comes at the cost of increasing execution time. This is true for all the thresholds except the last one, where execution increases and the solution quality gets worse. Why would allowing more of the available improvements to be implemented cause a reduction in quality? Because the methods do not implement non-improving moves, I am concluding that some improvements are damaging solution quality.

This pattern repeats across the no preference results. The R2 results, graph 7.1, are better than average example of the pattern, but the pattern exists with all methods apart from one. That method, No preference Partial Time (NPT), consistently produced the worst no preference solutions and did not contain this pattern. Because it produced such bad results the method does not look to be of value.

Some improvements look to have a damaging effect on the search process and should either be left till the end or just not implemented at all. We can see the reason for this by looking at the no preference results. The results show that the quality of the final solution was dependent on the acceptance threshold used. As the threshold percentile got lower more improvements could be found. What is interesting is that when the threshold was lowered further than the most effective threshold, so the last few improvements could be found, the final solution quality got worse. I expected it to be slower and produce the same quality of solution because of the extra time spent rejecting non-improving solutions, but it got worse. This means that using very low quality changes to locate improvements had a negative effect on quality.

If execution time allows, then it may be worth implementing these damaging improvements at the end of the search. Because using low quality moves to locate improvements is slow, it may not be worth spending extra time looking for them. Using very low quality changes to locate improvements is slow because they also locate a high proportion of non-improving moves.

The main observation here is that the location methods tested have an optimal threshold and using improvements below this threshold point had a damaging effect on final solution quality.

Low quality first works, but the initial threshold matters. The results show that giving preference to low quality changes improves final solution quality. The results also show that some low quality improvements have a damaging effect on final solution quality. We can see from the location experiments that if a lower than optimal threshold is used then it damages final solution quality. If an execution time of more than a few seconds is acceptable then the results suggest that using low quality first with a well-tuned initial threshold would work well.

The preference experiments start by using a threshold cut off that is lower than optimal identified in the location experiments. I suspect this is having a damaging effect on the results of the low quality first results. I suspect that the solution quality is being damaged and the positive effect of giving preference to low quality changes first is more significant than shown by the results. More experiments would need to be done to test if preference can have a significant positive impact when using partial changes.

The fact that some improvements have a damaging effect may explain why partial changes produce better quality solutions than whole. It is possible whole moves are less able to distinguish between the moves that are good to implement and improvements that cause further improvements to be more difficult to find. It is possible the cut off threshold for partial moves is excluding more of the damaging improvements than the cut off threshold for whole moves. I have little evidence if this is true or not but it is a possible explanation.

Typically, the amount of damage done to final solution quality by setting the threshold too low was between 0.5% and 2.5%. There were cases where the amount of damage was larger than this, in 2 cases the average damage was over 5%. See graph 7.1 for a typical example.

Comparing methods of locating improvements revealed the existence of damaging improvements. While this is not one of the things the experiments were setup to discover it does open up a potential area of research. Damaging improvements do exist and the method used to examine improvement location was able to suggest how to recognise them.

We can deal with damaging improvement through using preference or location methods. Implementing damaging improvements at the end of the search means we are giving preference to the improvements that have a less destructive effect. But if we choose not to implement the improvements at all, then there is no point in even trying to locate them.

The results in this section show that some very low quality solutions had a damaging effect on solutions quality. This suggests that an incorrectly tuned improvement location acceptance threshold can have a negative impact on both solution quality and execution time. The results here suggest that turning off the preference mechanism allows a good acceptance threshold to be identified.

7.4.2) Using low quality distance first to improve quality

This section reinforces the findings described in sections 7.3.1 and 7.3.2. It backs up the point that with all the VRPTW problems tested, giving preference to low quality distance tended to produce the best final quality solutions.

Giving preference to low quality distance moves gave the best quality final solution results for most problems, see tables 7.10 & 7.12. The search was guided by three different methods. These were distance, slack time and a combination of distance and slack time. When preference was given to using low quality distance moves to find improvements the search outperformed other whole move methods most of the time. Using low quality distance moves first gave the best quality whole move results for 8 of the 11 problems. With the 3 problems where other methods did better, the best quality results were achieved by a different method each time.

Using low quality whole or partial distance looks to almost always outperform the other methods when creating the best final solutions. The major exception to this is again C2_1, where LWD does not do very well. This problem style is possibly not that important because it is less realistic than random clustered (RC). RC is more

realistic because customers do typically exist in both cities and smaller towns and villages.

The gain in solution quality looks to have come at a cost of increasing the execution time needed. In the experiments giving preference to low quality distance first resulted in longer execution times. Giving preference to low quality distance first gave consistently lower quality solutions. After ½ a second giving preference to low quality distance first started to lag behind high quality first and no preference. It took low quality distance first several seconds to catch up and eventually overtake the other two methods.

With the experiments, giving preference to low quality moves improved final solution quality. This looks to be at the cost of doubling execution time.
Giving preference to high quality distance moves looked to make no difference to final solution quality with small time windows. The final solution quality was about the same as when no preference was given. With 4 problems it was a little better than giving no preference and in 5 problems a little worse.

Giving preference to high quality whole distance (HWD) looked to offer no improvements in speed in the very early stages of the search. After 0.5 seconds the solution quality of HWD lagged behind giving no preference (NWD) in every case. Again the scale of the lag could in theory be reduced with fine-tuning. High quality first did catch up and overtake after several seconds in some cases. In contrast high quality partial distance (HPD) dominates the large time window problems in the early stages of the search.

7.4.3) Longer execution time of low quality distance first

Giving preference to using Low quality Whole Distance moves, LWD, resulted in a better quality solution than giving preference to high quality, see tables 7.10 and 7.12. The gain in performance came at a cost of increasing the execution time needed. At the end of the 'hill climb' the solution quality of LWD was between 1.1%-4.8% better than HWD. Apart from C2_1, where low quality first was 0.7% worse than high quality first.

Giving preference to using low quality distance whole moves first also produced better final quality solutions than no preference with every problem style tested, again see tables 7.10 & 7.12.

The gain in solution quality that resulted from giving preference to LWD came at the expense of approximately doubling execution time.

When using partial changes to guide the search, there was a very notable difference in speed when giving preference to low quality changes first. Giving preference to low quality causes about a three and a half fold increase in execution time. At the start of the search this increase in execution time is much greater, taking more like ten times as long to reach a solution of similar quality. As the methods begin to reach a plateau the difference in execution time starts to reduce to 3.5 times as long.

Distinguishing between the improvement location and preference mechanisms allowed this trade off between solution quality and speed to be assessed. If the algorithms did not distinguish between the two mechanisms then it would be unclear which mechanism was having what impact.

7.4.4) Damaging improvements, the contradiction – Further work

There appears to be a contradiction between giving preference to low quality distance changes and the damaging improvements. Giving preference to low quality distance changes looks to improve final solution quality, see tables 7.10 & 7.12. As a result I would expect that improvements located using high quality changes would be the only damaging improvements. It looks as if there are damaging improvements at both ends of the scale. It is not clear from the results why this is the case.

The preference results contradict the idea that very high quality solutions have a damaging effect. When they give preference to high quality distance it does not appear to damage solution quality.

It appears further investigation of the impact of using very high quality improvements and very low quality improvements in the early stages of the search is required before damaging improvements can be understood.

7.4.5) Using high quality slack time first to improve quality

The results suggest that using whole move high quality slack time changes first, offers a benefit with larger problems. The same can not be claimed for partial moves because partial time methods did badly. With the 200 customer problems, using high

quality slack time changes first out performed low quality first. For all 5 of the 200 customer problems looked at, high quality slack time first was faster than low quality first, see tables 7.10 & 7.12.

There looks to be a cost in execution time with high quality slack time first. At the start of the search giving preference to high quality slack time lagged behind no preference to begin with. After one minute high quality slack time changes first had caught up with and overtaken no preference. High quality first was better in terms of time and quality than the other two methods after one minute.

The case for 100 customer problems was not so clear cut. High quality first produced the best quality solutions with 2 of the 6 problems. Which of the slack time methods performs best maybe be down to chance or it maybe down to problem type. Either way, there does not look to be a hard and fast rule that fits all of the 100 customer problem results.

7.4.6) Small reductions in slack time and distance improved quality.

When creating the best quality solutions using high quality whole slack time moves looks to offer an advantage over using low quality, see tables 7.10 & 7.12. This is not a contradiction to the idea that low quality change first produces better quality solutions. This is because distance is used to measure fitness and time is a constraint and thus not used to measure fitness. It is either feasible or unfeasible. This means my measure of slack time goes from lots of slack time through to negative slack time. Where negative slack means the change is unfeasible. A typical local search

method reduces both distance and route slack time. Reducing both of these by small amounts looks as if it will produce better quality solutions, although this appears to be at the cost of increasing execution time.

Giving preference to high quality time, and low quality distance, was not tested. The results suggest that such method could well outperform other preference methods on the VRPTW.

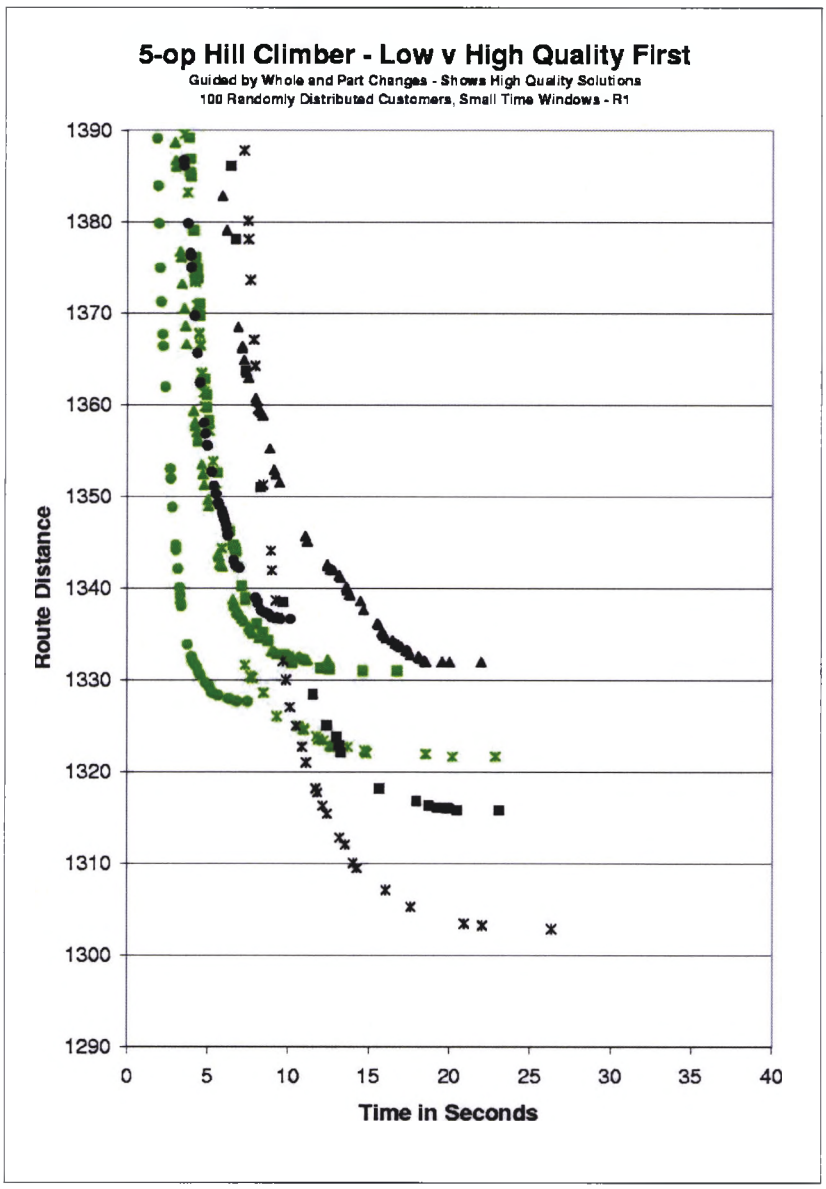
This section and section 7.4.5 highlight what can be learned by individually assessing the impact of different preference choices. Because preference mechanisms are separated from improvement location mechanisms it allows the impact of different preference mechanisms to be assessed. These sections describe how small reductions in the slackness of constraints (slack time) and solution fitness (distance) appear to improve final solution quality. The results described in these sections support the clam the distinguishing between location and preference is useful because doing so has shown which preference mechanisms have had a positive impact.

7.4.7) Designing an improvement preference

Giving preference to low quality improvements and small reductions in the tightness of the constraints produces the best quality solutions at the cost of increasing execution time, see Graph 7.2 and the early and best result Tables 7.4, 7.7, 7.10 & 7.12. The results indicate that very low quality solution changes should be avoided until the end of the search process and preference should be given to using lower quality changes to locate improvements.

While some improvements located using low quality changes look to damage the final solution quality, others look to have a positive effect. It looks to be down to where the cut off point is placed. This makes me wonder if the same is true for high quality changes. Maybe we should have two thresholds and not use moves below or above the thresholds until the end of the search. It is being suggested that an upper threshold may exist on the grounds that a lower threshold looks to exist. Although further investigation of this is required as these results provide little evidence one way or the other.

The results also suggest that giving preference to high quality distance typically speeds up the search but causes the search to tail off early. Giving preference to low quality distance and high quality slack time approximately doubles execution time while improving final solution quality.



Graph 7.2 – Preference results - speed v quality

Graph 7.2 description of key

- "Partial_Distance" - partial solution changes and the difference in distance were used to locate improvements.
- "Whole_Distance" - the method used whole previous solutions and the difference in distance to locate improvements.
- "Time" indicates the difference in solution slack time was used to locate improvements.
- "BestDistance+Time" - both the difference in distance and the difference in slack time were used to locate improvements.
- "High_First" (steep) - large reductions in slack time and/or distance were used to locate improvements before small reductions and increases were used.
- "Low_First" (shallow) - small reductions and increases were used to locate improvements before large reductions.

7.5) Locating improvements – general findings

On the whole, using the difference in distance between previous solutions proved to be a reliable method of locating improvements. Other factors appear to offer a small benefit. These are partial solution changes, acceptance threshold and using both distance and time. Details of all these improvement location methods are described next.

7.5.1) Partial distance changes more naturally suited to improvement location

Because the partial and whole move algorithms were both 5-opt hill climbers it was expected that they would produce the same quality final solutions, this did not happen. Partial distance changes typically produced better quality final solution than whole moves, see table 7.15 and tables 7.10 & 7.12. Although the significance of the

difference between the quality of the final solutions produced was small, see t-test table 7.11 & 7.13.

With no preference given to high or low quality changes							
	Clustered with Large Time Windows	Clustered with Small Time Windows	Random & Clustered with Large Time Windows	Random & Clustered with Small Time Windows	Random Distribution with Large Time Windows	Random Distribution with Small Time Windows	Average
How much partial changes were better than whole							
100 Customers	4.88%	4.93%	0.83%	0.68%	0.19%	0.38%	1.98%
200 Customers		4.28%	1.07%	1.19%	1.75%	1.01%	1.86%

Table 7.15 - How much partial changes were better than whole

The two hill climbers used slightly different methods of locating improvements. They also differed a little in how they decided which improvement to implement. There were two notable differences between the hill climbers: -

1. A whole move change locates a different set of improvements than a partial change. With both partial and whole moves, a single small change was used to locate several improvements. The best improvement out of the set of available improvements is then implemented. A partial change is used to locate improvements that share a single common edge. In contrast, a small whole move is used to locate improvements that share common edge(s) with the small move. This means that both methods are selecting the best improvement, but the sets they are selecting from are not the same. It is not known how often more than one improvement existed in a set of moves. The low quality first experiments

were done the opposite way round. With low quality first, the worst quality solution out of the set was implemented.

2. Improvements that did not get found: Both methods were designed so that almost all of the 2-5-opt improvements could be found. The difference was that the improvements missed by one method would not be the same as those missed by the other.

The two location mechanisms could have been designed so the 1st difference (1) did not exist. Instead both methods were designed to take advantage of their strengths. This resulted in the two differences described above.

The t-test p-values shows that the performance of these two methods is quite similar, but the fact that partial changes were able to produce better quality solutions with all 11 of the VRPTW problems suggests the pattern is not random. The reason why it is unclear which of the two differences caused the difference in performance is because the code did not allow the entire preference mechanism to be turned off, i.e. (1) above still chooses the best of the set of located improvements. Because of the damaging impact of very low quality improvement, see section 7.4.1, it would appear this was a bad design choice, this is because of the potential negative impact on the choice of acceptance threshold. If the acceptance threshold was impacted by this design choice then the difference between the high quality and low quality results could be less as a result.

Further work is needed to establish what impact these two reasons had on the difference in final solution quality. Although there is still a valuable observation to be made. The two different methods each locate a different natural set of improvements for implementation. The improvements located by partial changes produced slightly better quality final solutions. This was done with little processing time being spent on preference.

Needless to say, further work is needed on a wider range of problem types to confirm the generality of the positive findings due to partial solution changes.

7.5.2) Speed difference with whole moves and partial changes

When whole moves are compared to partial changes, the solution quality of partial change improves faster than that of whole move. Look at "Partial Distance Low First" and "Whole Distance Low First" in Graph 7.2 for a typical example. The percentage difference in solution quality increased as execution time increased. As the search progresses the number of improvement in the neighbourhood falls. This means it gets harder and harder to find improvements. As improvements became harder to find the benefits of using partial changes started to stand out more. Partial changes were faster at deciding a change would probably not result in an improvement being found. In short, using partial changes to locate hard to find improvements was notably more effective than using whole moves.

7.5.3) Locating improvements using distance, time and distance + time

This section links back to the use of high quality slack time in sections 7.4.5 and 7.4.6. Although the focus is on the contrast between the partial change and whole move results that were guided using slack time.

Four different methods of comparing solutions were used to locate improvements, Distance (D), Time (T), Distance + Time (DT) and Best Distance + Time (BDT) see table 7.2. Although two of them were very similar in design, DT and BDT. On the whole, using distance alone to locate improvements dominated. The exception to this is the fast solutions created for small time windows. These early solutions were created by methods that used both distance and slack time to guide the search, see tables 7.4 & 7.7.

The partial time (PT) methods look as if they could be improved upon. They always came bottom or near to the bottom in comparison with the other methods, see tables 7.4 to 7.13. The estimate of slack time was simplistic and looks as if it did not create a good estimate. The problem is that good methods of calculating the estimate look difficult to code and it remains unclear if they could be made to work. Although the partial solution change methods still make use of whole solutions and so can take advantage of using whole move slack time to guide the search.

The results described here suggest that accurate estimates of solution quality are required if partial solution changes are to be successful. Although the fact the partial change algorithms create whole solutions as well as partial solutions shows that

there is not a black and white distinction between the whole move and partial change mechanisms.

7.5.4) Finding the best tail off thresholds

This section and section 7.4.5 show that speed can be controlled using both acceptance thresholds and preference. By distinguishing between the mechanisms, the amount of difference that each mechanism makes to the execution speed can be evaluated.

Further speed gains look to be possible by reducing the acceptance thresholds. For each of the location methods 11 thresholds are compared, the result that is able to locate the largest improvement in solution quality is then used in the tables and my comparisons. The location results show that using only the best non-improving solutions to locate improvements speeds up the search process at the cost of reducing final solution quality. While the results show that both acceptance threshold and preference can be used to control speed, this study does not directly compare the two.

7.6) Method consistency and value of the results.

The experiments identify differences in solution quality and execution time with improvement location methods and preference styles. The use of partial and whole solutions was compared and methods of giving preference to high and low quality changes were compared with the aim of testing the hypothesis.

7.6.1) Comparison with other benchmark results

The 5-opt hill climbers look to produce lower quality solutions faster than what is typical for the Solomon benchmarks. The Table contains average solution qualities for the Solomon benchmark instances. The VRP solution qualities produced by [Bräysy 02] look to be some of the best available. The [Bräysy 02] solutions appear to have a total execution time of 82.5 minutes for the 56 instances, which would mean an average of 1.5 minutes per instance, although the paper does not appear to state if the 82.5min is a total or an average. The "Early Solutions" and "Best Solutions" are from tables 7.4-7.13 and show average execution times per instance.

The experiments limit improvement size to a 5-opt move. I suspect that increasing the size of the move operator will increase execution time and allow higher quality solution to be found, this is because sometimes more than 5 VRP edges to be changed to escape local optima.

	R1_1	R2_1	C1_1	C2_1	RC1_1	RC2_1	
Early Solutions	1482.33 *0.50sec*	*1173.56* *0.50sec*	1177.95 *0.50sec*	821.23 *0.48sec*	1587.40 0.49sec	1412.35 0.49sec	**:Distance** **:Time**
Best Solutions	1302.83 *26.34sec*	1007.64 *16.01sec*	907.02 *15.72sec*	700.26 *4.83sec*	1443.50 15.70sec	1203.87 10.72sec	**:Distance** **:Time**
[Bräysy 02]	1222.12	975.12	828.38	589.86	1389.58	1128.38	**:Distance** **82.5min**

Table 7.16 – Solomon benchmark results for 5-opt hill climber and [Braysy 02]

It is unclear if the lower that typical solution quality will have an impact on the usefulness of distinguishing between improvement location and preference. The experiment results do not show if distinguishing between improvement location and preference scales all the way up to very high quality solutions and so do not indicate

if distinguishing between the mechanisms still has value with such high quality solutions. The experiments use 5-opt hill climbers rather than variable size move operators or compound moves to compare location and preference mechanisms. Further experiments using variable size move operators or compound moves should tell us more about the value of the thesis hypothesis with very high quality solutions.

7.6.2) Number of Vehicles and Distance

Real VRP problems typically consider reducing the number of vehicles to be more important than reducing the distance. The results appear to have a close correlation between distance and the number of vehicles. This means that the distances compared are a good indication of the quality of the solution. This approach of comparing solution distance tends to be used in the literature [Cordeau 02]. This is unsurprising as the results given in this chapter show that as distance went down the number of vehicles also reduced.

7.7) Linking the results back to the hypothesis

The results show it is useful to distinguish between improvement location and improvement preference. The results are for the VRPTW and show which methods work well on which problems. The results suggest preference can be generally used to control the trade off between speed and solution quality, which is important, because different problems demand algorithms with different execution times. As for location, partial changes guided by distance tended to produce the best solutions with all problems except the early solution small time window problems, although the whole move distance method tended to follow closely behind.

The attached CD contains spreadsheets and source code. The spreadsheets contain all the raw results data and many graphs similar to Graphs 7.1 and 7.2. The source includes batch files that execute the code and produce the raw results files used in the spreadsheets.

A summary of the findings and an overview of what was learned are given in the conclusion chapter.

8) Conclusions

At the moment local search methods intertwine improvement location and improvement preference mechanisms. This thesis proposes that these mechanisms should be separated. The results show that explicitly separating improvement location and improvement preference mechanisms allows better understanding and control of the impact of each. The separation simplified the identification of location preference mechanisms that suited the VRPTW problems tested. The results also highlight how each mechanism can be adjusted to improve execution time or solution quality.

1. The results show the hypothesis, as stated and described in Chapter 4, is supported by experimental evidence: as shown in Chapter 7, distinguishing between improvement location and improvement preference proved feasible and produced useful results.
2. The principal mechanism of controlling the trade off between execution time and speed was improvement preference. See 7.4.2, 7.2 & 7.3.
3. The principal mechanism for further improving final solution quality was improvement location, guided by partial changes. See 7.5.1.

8.1) Criteria for success and overview of what was learned

The hypothesis states that: "Distinguishing between improvement location and improvement preference is feasible and useful when creating local search algorithms."

There appears to be little doubt that it is feasible, as the experiments described do distinguish between them. As regards usefulness, the results show that distinguishing between the two allowed the impact of each to be assessed, and

showed which of the mechanisms worked well together on which problems. This is useful because it removes part of the guesswork from the algorithm design process.

The following descriptions give an overview of what was learned and how the results differed from expectations: -

Slight preference in the location mechanism
The location mechanism used had embedded in it a slight preference for large improvements in solution quality, see section 7.5.1. This did not appear to cause problems with the results, i.e., there was a distinct difference between the speed and quality of the preference mechanisms, see 7.4.2 & 7.4.3. However the damaging improvements described in 7.4.1 suggest that if this slight preference that was embedded in the location mechanism had been turned off, the performance of the algorithms could have potentially been improved. The evidence from sections 7.4.1 – 7.4.3 & 7.5.1 suggests that one of the problems with not distinguishing between location and preference makes it difficult to work out which of the mechanism design choices are having a negative impact and which are having a positive impact.

The acceptance threshold is used by both mechanisms, because of this it can be easier to weaken the link between the mechanisms rather than actually separate them, but as noted in section 7.5.1 doing so appears to reduce the value of the results. Therefore when evaluating different location and preference mechanisms it is recommended that the time is taken to fully separate the mechanisms.

The damaging improvements

This is a strange result, as it feels counter intuitive that giving preference to low quality improvements, improves final solution quality, and yet, giving preference to extremely low quality improvements damages solution quality. On the surface this looks to be a difficult problem to get to the bottom of, because it appears to involve compound moves. Exploring the impact of compound moves can demand a large amount of computational processing power as it involves making quite a few changes to the solution.

Impact of partial change

The expected impact of partial changes was less than I expected. Although [Johnson 97] does point out that the difference between using whole changes and Lin-Kernighan partial changes was only a few percent on the TSP.

Development time

The major extra cost of distinguishing between preference and location methods looks to be development time. The experience, of developing the partial changes algorithms, suggests the more complex mechanisms, such as partial changes and using many solutions to help locate improvements, are difficult to implement and demand extra development time. The development time was probably about 50% longer than would be required if little distinction between the mechanisms was made. Johnson [Johnson 97] also points out that Lin-Kernighan which uses partial moves is difficult to implement. The extra development time needed for the whole move algorithms was less extreme, with them taking around 20% longer to develop.

Preference and execution time
The actual impact of preference was unexpected, it made a big difference to execution time, in 0.5 seconds high quality first improved initial solution quality by around 50% with the large time window problems. Low quality first improved solution quality by less than 2% with the same amount of execution time.

Difference between solutions and preference
It was assumed small reductions in slack time would have the same impact as large increases in solution quality. The results show the reverse is probably true. The results suggest small reductions in slack time and small increases in solution quality produce better quality final solutions. The invalid assumption probably led to some of the improvement location mechanisms being badly designed. The mechanisms that were guided by both slack time and distance could probably have been designed better if the assumption has not been made. This indicates that independently testing each of the guidance variables, such as distance (fitness) and slack time (constraints) before attempting to combine them together is worth doing. Because they could be pulling in opposite directions, independently testing their impact on the location and preference mechanisms should help show what impact they have on solution quality and execution time.

How much does execution time matter?
For problems such as the VRPTW, the odds are that execution times of 10mins or even 30mins are fine. This means the best early solution methods, that started to tail off in around 0.5 seconds, for both 100 and 200 customer problems, look to be of little use for VRPTW problems with less than 200 customers. This is because the slower methods that produced higher quality solutions all tailed off in between 25 seconds and 25 minutes, see Tables of Best quality solutions. The point is that

execution times do matter, but it is the scale of execution time that matters most. With some problems an execution time of several seconds maybe too long and yet with others an execution time a several days is acceptable.

Understanding local search

In practical terms dividing the local search methods up into methods of locating improvements and improvement preference makes the concepts of local search easy to understand. It is complicated to describe a method such as Tabu Search, whereas, in comparison, it is simpler to understand the concept of locating an improvement. People can relate to the fact that there are different methods of locating improvement and that some will work better than others. Also, the concept of giving preference to one improvement rather than another is simple to relate to. The results show that separating location and preference teaches us about how local search works and in terms simpler to understand and more useful than say TS or GAs.

8.2) Summary of Contributions

Practical Contribution to knowledge
By separating the improvement location and improvement preference mechanisms, the following was learned: -

Quality	Speed	What the VRPTW results highlighted
Y		Giving preference to lower quality solutions improved the quality of the final solution. See 7.3, 7.4.2 & 7.4.3
	Y	Giving preference to higher quality solutions increased the speed of the algorithm at the cost of slightly reducing quality with the large time window problems. See 7.2.1 & 7.4.2
	Y	When the problem had tight constraints and fast execution times were needed using both constraints and solution quality to guide the search worked best. See 7.2.2
Y		Gradually tightening constraints improved final solution quality. E.g. gradually reducing the amount of slack time available in the delivery route improved final solution quality. See 7.4.6
Y		Partial solution changes produced slightly better quality solutions than using whole moves. See 7.5.1
Y	Y	Disabling the improvement preference mechanism allowed the identification of a good acceptance threshold, below which few improvements existed. These improvements had a negative impact on the final solution quality. Identifying a good acceptance threshold both reduced execution time and improved solution quality. See 7.4.1
Y	Y	The more accurate estimates of the value of incomplete solutions vastly outperformed the rough estimates, in terms of both speed and solution quality. See 7.5.3

General contribution to knowledge
Asking the following question should help identify any exceptions: "What information does the method use to help find a new best solution?"

So far, it looks as if the answer can be pinned down to a single variable created using "the difference in solution quality between two or more solutions".

Considerable effort is sometimes needed to trace decisions back to this variable. For example look at how an Ant Colony method finds a new best solution to a TSP problem. It uses the weights of the edges to help it decide which edges to use to help locate a new best solution. Take a simplified version of the method that uses only two previous solutions to help find a new best solution. It can be seen that when choosing between two edges it is the amount of difference between the solutions that determines the probability that a particular edge will be used and therefore the probability a particular improvement will be found. Actual Ant Colony methods use several solutions to help find a new best solution rather than just a single pair of solutions.

From the cases examined in the literature review it appears the difference in solution quality between two or more solutions is consistently being used to locate improvements and to make preference choices. However, far from all local search methods have been examined here.

It should be pointed out that several independent variables are also used when locating improvements, such as the move operator, move operator size, and preference. However the difference in solution quality appears to be the main, if not the only, dependent one.

8.3) Future Research – improvement location mechanisms

Using many whole solutions
There is a sliding scale from using a partial solution change to locate improvements to using many solutions. The experiments only looked at one end of this scale and showed that partial solutions only produced the best solutions on some problem styles and performed less well on others. Examination of the entire sliding scale looks as if it would tell us more about which methods work with which problem types.

Construction heuristics
Construction heuristics may well be able to take advantage of the ideas presented here, as they also employ improvement location mechanisms and improvement preference mechanisms.

8.4) Future Research – General

Non-changing threshold
The local search methods ability to distinguish between improving moves and non-improving moves did not change during the hill climb process. This principle has implications for the choice of cut off threshold. If the principle is false then the task of identifying a good cut off threshold is made more complicated. This is because if the ability to distinguish between improving moves and non-improving changes as the search progresses then the threshold should also change. However, the formative results suggest the ability to distinguish does not change.

Variable and fix size move operators

To locate improvements, meta-heuristics use fixed size move operators or variable size move operators. The experiments only tested the use of a limited variable size move operator, which is only able to locate 2, 3, 4 & 5-opt improvements.

Damaging effect of very low quality solutions
Using very low quality solutions to locate improvements had a damaging effect on final solution quality. While the results showed this was taking place, because it was unexpected the experiments did not gather information to try and understand why and when this happens.

8.5) Future Research – other problem types

Partial changes
The experiments show it is feasible to use partial solutions to locate improvements and make preference choices on the VRPTW. The experiments also show that using the difference in quality between partial solutions offers a better estimate of the value of a solution property on the VRPTW, see sections 7.5.1.& 7.5.3. The use of partial solutions on problems other than the VRPTW was not explored. Because greedy construction methods typically assess the value of partial solutions when constructing solutions it looks as if it is feasible to use partial solutions to guide improvement location and preference on problems other than the VRPTW.

Combinatorial problems come in many forms, scheduling, vehicle routing, time tabling etc. Many formulas for calculating the quality of a whole solution for these problems exist. With a partial solution we can not test if all constraints can be satisfied. The results described in section 7.5.3 suggest that calculation of partial solution quality needs to take into account possible constraint violations when estimating solution quality. There is probably a lot more to be learned about

evaluating partial solutions. It is still unclear how many problems can successfully use partial solutions, but the results suggest further experiments on other problem types are worth doing.

Location and preference mechanisms
Because these mechanisms exist in meta-heuristics already it is suspected that distinguishing between the two on other problems will produce usable information. There is still much to be learned about which mechanisms work best with the VRPTW and other problems. The fact that there were few difficulties separating the mechanisms in the experiments it suggests the separating the mechanisms on other problem types will not be difficult to do.

8.6) Implications for local search meta-heuristics – further work

Judging from the results of the VRPTW experiments there looks to be several ways existing local search meta-heuristics can be improved. The experiments only produced results for VRPTW problems and it remains unclear if the findings can be successfully applied to other combinatorial optimisation problems.

Some of the ideas are simple to test out on other problems and some are much more complex. As a result it may be worth testing out the easy to implement ideas on existing code. In cases where a major re-write would be needed, the argument that the idea is worth trying is less convincing. This is because the ideas have not been tried on other problems and therefore a degree of caution is necessary.

8.6.1) Tabu Search - improvement preference

The results suggest solution quality can be improved by giving preference to low quality solution changes. This is because both Low quality Whole Distance (LWD) and Low quality Partial Distance (LPD) typically produced the highest quality solutions across all the VRPTW problem types. Tabu Search typically gives preference to high quality solution changes. Glover [Glover 97] recommends the use of steepest ascent or candidate lists. The results described in section 7.4.2 suggest steepest ascent and candidate lists do not produce the best quality final solutions for the VRPTW. The results also show that giving preference to High quality (steep ascent) Whole Distance first (HWD), performed about as well as No preference (any ascent) Whole Distance (NWD). This suggests that using steepest ascent whole moves is not worth doing with VRPTW.

Giving preference to low quality increases execution time. If an increase in execution time is acceptable then changing existing code to give preference to low quality is probably worth trying. The easiest change to make is probably changing steepest ascent to smallest ascent. While I do not compare steepest ascent with smallest ascent, I suspect this will improve solution quality at the cost of execution time. Although because Tabu Search continues to look for improvements after the hill climb is complete, it is hard to estimate the impact on execution time and solution quality.

Judging by the results, the amount of difference in quality will probably be quite small in most cases.

8.6.2) Simulated Annealing and threshold methods - improvement preference

Again, the VRTPW results suggest solution quality can be improved giving preference to low quality solution changes. Simulated Annealing accepts all improving moves as and when they are found. It looks like it could be possible to further improve final solution quality by giving preference to low quality improvements at the start of the search. It should be noted, that doing this, may more than double execution time. I am assuming this because, giving preference to low quality whole moves more than doubled execution time in the experiments in this study.

The same arguments apply to threshold methods. Threshold methods implement all improvements as and when they are found. This means they could also potentially benefit from giving preference to low quality improvements at the start of the search.

8.6.3) Evolutionary Algorithms - improvement preference

While changes to the improvement preference may improve Evolutionary Algorithms the evidence one way or the other is very thin. I did not do experiments to test the use of many solutions to guide the improvement preference. Judging by the VRPTW results obtained, it looks like giving preference to high quality improvements offers no benefit to final solution quality. The problem is the experiments use a small number of property changes to locate improvements. Evolutionary Algorithms can locate improvements using a large number of property changes.

If using many solutions to guide the search is like using either pairs of whole move or partial changes, then giving preference to high quality whole solutions has a potential impact on solution quality and execution time. Evolutionary Algorithms tend to give preference to high quality whole solutions, rejecting members of the population with a low solution quality. The VRPTW results suggest this is worth doing if solutions need to be created within a few seconds. That said the evidence one way or the other is a little weak because using more than one solution to locate an improvement was not tested here.

Giving preference to solutions that are only a little better than the solutions used to create them may offer a benefit with VRPTW and similar problems. Again the results suggest this would both improve final solution quality and increase execution time. This is probably the easiest change to make to an existing EA. As with high quality first, the impact on solution quality is hard to estimate.

It is hard to suggest how to make improvements to Evolutionary Algorithms without further experiments. The hypothesis argues that methods of using many solutions to locate improvements should be compared. It also argues that preference styles should be compared. Without such experiments it is hard to understand the benefits of using many solutions to locate improvement

8.7) Summary of what has been covered

Local search heuristics that use non-improving solutions can be seen as hill climbers that use the non-improving solutions to help them find improvements. Local search methods use these non-improving solutions in slightly different ways. Most local search meta-heuristics make use of non-improving solutions to help locate improvements: -

- Simulated Annealing uses non-improving solutions biasing its choice towards solutions that are only a little worse.
- Tabu Search uses the best non-improving solutions in the neighbourhood.
- EAs maintain a neighbourhood (population) of non-improving solutions, as new solutions are created weaker solutions are discarded.
- Ant Colonies use a neighbourhood (population) of whole solutions biased towards more recent solutions.

Two mechanisms that exist in local search methods have been highlighted: first is the mechanism they use to locate improvements; and second is the mechanism used to give preference to using some improvements rather than others. Current meta-heuristics merge the measurement both of these mechanisms using a single performance measure. The experimental results show much can be learned by independently measuring improvement location and preference.

9) Appendix - Standard Deviations

Early Solutions, Large Time Windows

Standard Deviations from 40 runs

Time in seconds

C2_1	Time 0.5	Rc2_1	Time 0.5	Rc2_2	Time 0.5	R2_1	Time 0.5	R2_2	Time 0.5
FPD	153	FPD	315	FPD	835	FPD	348	FPD	1803
FWD	114	FWD	227	FWD	375	FWD	211	FWD	381
FPT	224	FPT	230	FPT	619	FPT	228	FPT	512
FWT	147	FWT	148	FWT	474	FWT	138	FWT	291
FPDT	194	FPDT	461	FPDT	1278	FPDT	621	FPDT	2333
FWDT	109	FWDT	172	FWDT	424	FWDT	151	FWDT	393
FPBDT	277	FPBDT	406	FPBDT	532	FPBDT	539	FPBDT	1892
FWBDT	95	FWBDT	156	FWBDT	411	FWBDT	133	FWBDT	369
HPD	175	HPD	346	HPD	785	HPD	270	HPD	1635
HWD	131	HWD	219	HWD	411	HWD	236	HWD	536
HPT	306	HPT	211	HPT	650	HPT	146	HPT	430
HWT	225	HWT	178	HWT	621	HWT	137	HWT	383
HPBDT	306	HPBDT	269	HPBDT	650	HPBDT	146	HPBDT	430
HWBDT	220	HWBDT	145	HWBDT	618	HWBDT	123	HWBDT	358
LPD	306	LPD	211	LPD	650	LPD	155	LPD	427
LWD	330	LWD	192	LWD	662	LWD	124	LWD	517
LPT	306	LPT	187	LPT	650	LPT	152	LPT	438
LWT	242	LWT	180	LWT	584	LWT	124	LWT	457
LPBDT	287	LPBDT	260	LPBDT	650	LPBDT	230	LPBDT	480
LWBDT	208	LWBDT	154	LWBDT	639	LWBDT	128	LWBDT	401

Early Solutions, Small Time Windows

Standard Deviations from 40 runs

Time in seconds

C1_1	Time 0.5	C1_2	Time 0.5	Rc1_1	Time 0.5	Rc1_2	Time 0.5	R1_1	Time 0.5	R1_2	Time 0.5
FPD	208	FPD	799	FPD	306	FPD	806	FPD	397	FPD	1166
FWD	168	FWD	661	FWD	210	FWD	452	FWD	267	FWD	461
FPT	221	FPT	690	FPT	168	FPT	513	FPT	203	FPT	528
FWT	140	FWT	479	FWT	182	FWT	332	FWT	210	FWT	522
FPDT	239	FPDT	1208	FPDT	301	FPDT	996	FPDT	462	FPDT	1378
FWDT	131	FWDT	389	FWDT	217	FWDT	321	FWDT	236	FWDT	480
FPBDT	219	FPBDT	1319	FPBDT	367	FPBDT	760	FPBDT	397	FPBDT	932
FWBDT	116	FWBDT	308	FWBDT	193	FWBDT	298	FWBDT	220	FWBDT	501
HPD	241	HPD	1454	HPD	330	HPD	755	HPD	408	HPD	945
HWD	157	HWD	1124	HWD	229	HWD	481	HWD	266	HWD	548
HPT	230	HPT	705	HPT	191	HPT	524	HPT	177	HPT	441
HWT	175	HWT	682	HWT	175	HWT	436	HWT	186	HWT	364
HPBDT	230	HPBDT	705	HPBDT	240	HPBDT	534	HPBDT	296	HPBDT	441
HWBDT	160	HWBDT	757	HWBDT	194	HWBDT	468	HWBDT	194	HWBDT	367
LPD	231	LPD	705	LPD	193	LPD	537	LPD	176	LPD	441
LWD	197	LWD	649	LWD	173	LWD	548	LWD	154	LWD	478
LPT	216	LPT	705	LPT	180	LPT	534	LPT	161	LPT	441
LWT	181	LWT	631	LWT	158	LWT	441	LWT	153	LWT	423
LPBDT	221	LPBDT	802	LPBDT	248	LPBDT	534	LPBDT	191	LPBDT	443
LWBDT	200	LWBDT	576	LWBDT	153	LWBDT	464	LWBDT	165	LWBDT	408

Point early solutions are overtaken, Large Time Windows

Standard Deviations from 40 runs

Time in seconds

C2_1	Time 30	Rc2_1	Time 10	Rc2_2	Time 150	R2_1	Time 11	R2_2	Time 50
FPD	64	FPD	195	FPD	294	FPD	170	FPD	618
FWD	75	FWD	187	FWD	340	FWD	167	FWD	628
FPT	65	FPT	213	FPT	367	FPT	392	FPT	2088
FWT	59	FWT	195	FWT	314	FWT	155	FWT	558
FPDT	53	FPDT	228	FPDT	321	FPDT	347	FPDT	1545
FWDT	71	FWDT	177	FWDT	280	FWDT	159	FWDT	631
FPBDT	59	FPBDT	189	FPBDT	356	FPBDT	345	FPBDT	1560
FWBDT	72	FWBDT	179	FWBDT	367	FWBDT	166	FWBDT	608
HPD	58	HPD	189	HPD	303	HPD	169	HPD	617
HWD	76	HWD	183	HWD	309	HWD	166	HWD	645
HPT	181	HPT	433	HPT	1331	HPT	417	HPT	1589
HWT	83	HWT	194	HWT	317	HWT	145	HWT	600
HPBDT	67	HPBDT	333	HPBDT	322	HPBDT	401	HPBDT	1794
HWBDT	64	HWBDT	207	HWBDT	332	HWBDT	159	HWBDT	657
LPD	128	LPD	239	LPD	1298	LPD	178	LPD	1271
LWD	51	LWD	169	LWD	300	LWD	103	LWD	478
LPT	216	LPT	210	LPT	747	LPT	239	LPT	1105
LWT	60	LWT	178	LWT	315	LWT	159	LWT	539
LPBDT	56	LPBDT	202	LPBDT	295	LPBDT	315	LPBDT	1432
LWBDT	68	LWBDT	165	LWBDT	311	LWBDT	162	LWBDT	606

Point early solutions are overtaken, Small Time Windows

Standard Deviations from 40 runs

Time in seconds

C1_1	Time 4	C1_2	Time 50	Rc1_1	Time 12	Rc1_2	Time 38	R1_1	Time 9	R1_2	Time 40
FPD	87	FPD	202	FPD	200	FPD	415	FPD	231	FPD	840
FWD	83	FWD	191	FWD	196	FWD	446	FWD	241	FWD	887
FPT	173	FPT	240	FPT	202	FPT	446	FPT	316	FPT	2193
FWT	92	FWT	196	FWT	207	FWT	370	FWT	230	FWT	784
FPDT	91	FPDT	213	FPDT	234	FPDT	425	FPDT	333	FPDT	2139
FWDT	61	FWDT	203	FWDT	205	FWDT	401	FWDT	231	FWDT	830
FPBDT	84	FPBDT	167	FPBDT	215	FPBDT	411	FPBDT	332	FPBDT	2085
FWBDT	87	FWBDT	222	FWBDT	210	FWBDT	435	FWBDT	238	FWBDT	882
HPD	119	HPD	188	HPD	204	HPD	405	HPD	239	HPD	815
HWD	111	HWD	220	HWD	193	HWD	415	HWD	231	HWD	843
HPT	268	HPT	2687	HPT	329	HPT	1553	HPT	377	HPT	1484
HWT	102	HWT	228	HWT	196	HWT	295	HWT	232	HWT	759
HPBDT	332	HPBDT	2858	HPBDT	281	HPBDT	1532	HPBDT	444	HPBDT	2050
HWBDT	90	HWBDT	187	HWBDT	208	HWBDT	409	HWBDT	232	HWBDT	856
LPD	247	LPD	2087	LPD	234	LPD	534	LPD	283	LPD	605
LWD	106	LWD	214	LWD	197	LWD	208	LWD	232	LWD	574
LPT	183	LPT	799	LPT	252	LPT	568	LPT	166	LPT	542
LWT	118	LWT	189	LWT	194	LWT	410	LWT	225	LWT	732
LPBDT	113	LPBDT	235	LPBDT	224	LPBDT	451	LPBDT	334	LPBDT	1767
LWBDT	93	LWBDT	206	LWBDT	203	LWBDT	404	LWBDT	236	LWBDT	770

Best quality solutions, Large Time Windows

Standard Deviations from 40 runs

Time in seconds

C2_1	Time 50	Rc2_1	Time 50	Rc2_2	Time 1300	R2_1	Time 50	R2_2	Time 164
FPD	64	FPD	195	FPD	294	FPD	170	FPD	618
FWD	75	FWD	187	FWD	340	FWD	168	FWD	628
FPT	68	FPT	199	FPT	361	FPT	332	FPT	1794
FWT	59	FWT	198	FWT	319	FWT	163	FWT	577
FPDT	53	FPDT	228	FPDT	321	FPDT	337	FPDT	1484
FWDT	71	FWDT	177	FWDT	280	FWDT	159	FWDT	631
FPBDT	59	FPBDT	189	FPBDT	356	FPBDT	345	FPBDT	1554
FWBDT	72	FWBDT	179	FWBDT	367	FWBDT	166	FWBDT	608
HPD	58	HPD	189	HPD	303	HPD	169	HPD	617
HWD	76	HWD	182	HWD	310	HWD	164	HWD	645
HPT	216	HPT	315	HPT	371	HPT	372	HPT	1725
HWT	83	HWT	205	HWT	322	HWT	154	HWT	621
HPBDT	67	HPBDT	212	HPBDT	300	HPBDT	315	HPBDT	1531
HWBDT	64	HWBDT	208	HWBDT	332	HWBDT	160	HWBDT	657
LPD	50	LPD	206	LPD	335	LPD	156	LPD	578
LWD	51	LWD	195	LWD	343	LWD	163	LWD	643
LPT	176	LPT	260	LPT	625	LPT	251	LPT	1273
LWT	58	LWT	186	LWT	348	LWT	165	LWT	561
LPBDT	56	LPBDT	202	LPBDT	295	LPBDT	309	LPBDT	1432
LWBDT	68	LWBDT	182	LWBDT	321	LWBDT	166	LWBDT	607

Best quality solutions, Small Time Windows

Standard Deviations from 40 runs

Time in seconds

C1_1	Time 50	C1_2	Time 420	Rc1_1	Time 50	Rc1_2	Time 240	R1_1	Time 25	R1_2	Time 600
FPD	76	FPD	188	FPD	194	FPD	398	FPD	224	FPD	819
FWD	109	FWD	191	FWD	196	FWD	404	FWD	239	FWD	830
FPT	81	FPT	219	FPT	224	FPT	400	FPT	323	FPT	1603
FWT	99	FWT	211	FWT	207	FWT	387	FWT	231	FWT	819
FPDT	92	FPDT	213	FPDT	234	FPDT	425	FPDT	320	FPDT	1357
FWDT	61	FWDT	203	FWDT	205	FWDT	401	FWDT	231	FWDT	830
FPBDT	73	FPBDT	167	FPBDT	212	FPBDT	411	FPBDT	323	FPBDT	1359
FWBDT	87	FWBDT	222	FWBDT	210	FWBDT	436	FWBDT	238	FWBDT	882
HPD	67	HPD	187	HPD	204	HPD	406	HPD	228	HPD	789
HWD	96	HWD	219	HWD	193	HWD	406	HWD	228	HWD	825
HPT	255	HPT	1629	HPT	239	HPT	1411	HPT	345	HPT	1713
HWT	79	HWT	250	HWT	196	HWT	349	HWT	232	HWT	839
HPBDT	85	HPBDT	210	HPBDT	219	HPBDT	418	HPBDT	335	HPBDT	1256
HWBDT	92	HWBDT	187	HWBDT	208	HWBDT	420	HWBDT	232	HWBDT	861
LPD	104	LPD	167	LPD	214	LPD	387	LPD	232	LPD	809
LWD	75	LWD	184	LWD	200	LWD	408	LWD	231	LWD	838
LPT	249	LPT	1641	LPT	237	LPT	994	LPT	174	LPT	860
LWT	80	LWT	193	LWT	194	LWT	373	LWT	228	LWT	824
LPBDT	73	LPBDT	155	LPBDT	224	LPBDT	418	LPBDT	319	LPBDT	1217
LWBDT	84	LWBDT	213	LWBDT	203	LWBDT	410	LWBDT	236	LWBDT	810

10) References

[Anderson 96] Theory and Methodology: Mechanisms for local search – European Journal of Operational Research 88, pages 139-151 – E.J. Anderson, 1996

[Applegate 99] Finding tours in the TSP – Tech. Rep. TR99-05, Departement of Computational and Applied Mathematics, Rice University – D. Applegate, R. Bixby, V. Chvátal, W. Cook, 1999 – http://www.tsp.gatech.edu/papers/

[Bäck 96] Evolutionary Algorithms in Theory and Practice – Oxford University Press – T. Back, 1996

[Bäck 97] Handbook of Evolutionary Computation – IOP Publishing Ltd and Oxford University Press – T. Bäck, D. B. Fogel, and Z. Michalewicz.1997.

[Bentley 90] Experiments on Traveling Salesman Heuristics – in 1st Annual ACM-SIAM Symp. Discrete Alg. (SODA '90), pages 91--99, San Francisco, editor: D. S. Johnson – J. L. Bentley, 1990

[Bierwirth 96] On permutation representations for scheduling problems – In Parallel Problem Solving from Nature, PPSN IV, pages 310-- 318, Berlin, Springer, editors: Hans-Michael Voigt, Werner Ebeling, Ingo Rechenberg, and Hans-Paul Schwefel – Christian Bierwirth, Dirk C. Mattfeld, and Herbert Kopfer 1996

[Braun 99] A Comparison Study of Static Mapping Heuristics for a Class of Metatasks on Heterogeneous Computing Systems – In Proceedings of the 8th Heterogeneous Computing Workshop (HCW'99), pages 15-29, – R.D Braun, H.J. Siegel, N. Beck, L. Boloni, M. Maheswaran, A.I. Reuther, J.P. Robertson, M.D. Theys, B. Yao, D. Hensgen, and R.F. Freund Apr. 1999

[Bräysy 02] Tabu Search Heuristics for the Vehicle Routing Problem with Time Windows. – Top 10:2, 211-238, the journal of Spanish operations research society – O. Bräysy, and M. Gendreau, 2002 – http://www.sintef.no/static/am/opti/projects/top/publications.html

[Bremermann 62] Optimization through evolution and recombination – in Sels-organizing systems, Washington, Spartan Books, editors: M. C. Yovits et al. 93-106 – H. J. Bremermann 1962

[Congram 98] An iterated dynasearch algorithm for the single--machine total weighted tardiness scheduling problem – Technical report, Faculty of Mathematical Studies, University of Southampton – R. K. Congram, C. N. Potts, and S. L. Van de Velde, 1998

[Cordeau 02] A Guide to Vehicle Routing Heuristics – in Journal of the Operational Research Society 53:512-522 – J.-F. Cordeau, M. Gendreau, G. Laporte, J.-Y. Potvin and F. Semet, 2002

[Cordone 01] A Heuristic Algorithm for the Vehicle Routing Problem with Time Windows – in Journal of Heuristics, 7(2), March 2001 – R. Cordone, R. Wolfler Calvo, 2001

[Corne 96] Peckish initialisation strategies for evolutionary timetabling – in, Practice and Theory of Automated Timetabling p227-240 – D Corne, P Ross 1996

[Corne 99] New Ideas in Optimization – McGraw-Hill, London – D. Corne, M. Dorigo, F. Glover. (ed.), ISBN: 0077095065, 1999

[DeJong 75] An analysis of the behavior of a class of genetic adaptive systems – PhD thesis, University of Michigan – K. A. De Jong 1975

[Dorigo 97] Ant colony system: A cooperative learning approach to the traveling salesman problem – in IEEE Transactions on Evolutionary Computation, 1(1):53—66 – M. Dorigo and L. Gambardella 1997.

[Dorigo 99] The ant colony optimization meta-heuristic" – in New ideas in optimization D. Corne, M. Dorigo, and F. Glover editors – M Dorigo, G Di Caro 1999

[Dorn 95] Case-based reactive scheduling – in Artificial Intelligence in Reactive Scheduling – London: Chapman & Hall – Roger Kerr and Elisabeth Szelke (eds) – pp. 32-50 – Dorn, J 1995.

[Dowsland 93] Simulated Annealing – in Modern Heuristic Techniques for Combinatorial Problems C.Reeves (editor) – K. Dowsland 1993

[Dueck 90] Threshold Accepting: A General Purpose Optimization Algorithm Superior to Simulated Annealing – in Journal of Computational Physics, 90:161—175 – Gunter Dueck and Tobias Scheuer 1990

[Dueck 93] New optimization heuristics: The great deluge algorithm and the record-torecord -travel . Journal of Computational Physics, 104(1):86--92, also available as technical report TR 89.06.11, IBM Germany, Heidelberg Scientific Center – Gunter Dueck 1993

[Fogel 66] Artificial intelligence through simulated evolution. Wiley, New York. L. J. Fogel, A. J. Owens and M. J. Walsh 1966

[Fogel 95] Evolutionary Computation. Toward a New Philosophy of Machine Intelligence, IEEE Press – D. Fogel 1995

[Garey 79] Computers and Intractability, A Guide to the Theory of NP-Completeness. W.H. Freeman and Company, New York – M.R. Garey and D. S. Johnson 1979

[Glover 97] Tabu Search – Kluwer Academic Publishers – F. Glover, M. Laguna 1997

[Goldberg 87] Genetic algorithms with sharing for multimodal function optimization. In J. J. Grefenstette, editor, Proceedings of the Second International Conference on Genetic Algorithms, pages 148--154. San Francisco, CA: Morgan Kaufmann – D. E. Goldberg and J. Richardson 1987

[Goldberg 89] Genetic Algorithms in Search, Optimization, and Machine Learning – Addison-Wesley – D. E. Goldberg 1989

[Held 70] The traveling-salesman problem and minimum spanning trees. Operations Research, 18:1138--1162 – M. Held and R.M. Karp. 1970

[Held 71] The traveling salesman problem and minimum spanning trees: part II, Mathematical Programming 1 pp. 6-25 – M. Held and R.M. Karp 1971

[Holland 75] Adaptation in Natural and Artificial Systems. Ann Arbor: The University of Michigan Press – J. H. Holland 1975

[Johnson 96] Asymptotic experimental analysis for the Held-Karp Traveling Salesman bound. In Proceedings of the 7th Annual ACM-SIAM Symposium on Discrete Algorithms, pages 341-350 – D. S. Johnson, L. A. McGeoch, and E. E. Roghberg. 1996

[Johnson 97] The Travelling Salesman Problem: A Case Study in Local Optimization – D. Johnson and L. A. McGeoch 1997 in Local Search in Combinatorial Optimization – John Wiley and Sons – Aarts and Lenstra (eds) – pp. 215-310 – 1997

[Johnson 02] Experimental analysis of heuristics for the STSP – to appear in The Travelling Salesman Problem and its Variations – D Johnson, L McGeoch 2002 – TSP Challenge http://www.research.att.com/~dsj/chtsp/

[Juels 94] Stochastic Hill-climbing as a Baseline Method for Evaluating Genetic Algorithms, Technical Report, University of California at Berkeley – A. Juels , M. Wattenberg 1994 – also in: D. S. Touretzky, M. C. Mozer, and M. E. Hasselmo, editors, Advances in Neural Information Processing Systems, volume 8, pages 430--436. 1996

[Kirkpatrick 83] "Optimization by simulated annealing," Science, vol. 220, pp. 671—680 – S. Kirkpatrick, C. D. Gellatt, J. Vecchi, and M. P. Vecchi 1983.

[Lin 73] – An Effective Heuristics Algorithm for the Traveling Salesman Problem – in Operations Research 21 pp. 498-516 – S. Lin & B. W. Kernighan 1973

[Lorenco 02] Iterated Local Search – in Handbook of metaheuristics, F. Glover and G. Kochenberger, editors – H.R. Lorenzo, O. Martin, and T. Stuetzle 2002

[Madsen 98] A New Branch-and-Bound Method for Global Optimization – IMM-REP-1998-05, Department of Mathematical Modeling, Technical University of Denmark, DK-2800 Lyngby, Denmark – Kaj Madsen and Serguei Zertchaninov 1998 http://citeseer.ist.psu.edu/madsen98new.html

[Martin 91] Large-step markov chains for the traveling salesman problem – Complex Systems 5. p299-326. O. Martin, S. Otto, and E. Felten, 1991.

[Michalewicz 02] How to Solve It: Modern Heuristics – Springer Verlag – Z. Michalewicz and D.B. Fogel 2002.

[Ozdamar 02] New results for the capacitated lot sizing problem with overtime decisions and setup times – Production-Planning-and-Control. vol.13, no.1; Jan.-Feb. p.2-10. L. Ozdamar, P.Y. Birbil, M.C. Portmann 2002

[Papadimitriou 82] Combinatorial Optimization: Algorithms and Complexity – Prentice-Hall – C. H. Papadimitriou and K. Stiglitz 1982.

[Poupaert 01] Acceptance driven selection: an approach to approximate global search strategies in local search and evolutionary algorithms. – Proceedings of the Genetic and Evolutionary Computation Conference pp 1173-1180 Morgan Kaufmann – E. Poupaert, Y. Deville 2001

[Rechenberg 73] Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution, Fromann-Holzboog – I. Rechenberg 1973

[Reeves 93] Modern Heuristic Techniques for Combinatorial Problems – Blackwell Scientific Publications, Oxford, C.Reeves editor 1993

[Rochat 94] A tabu search approach for delivering pet food and flour in Switzerland – Journal of the Operational Research Society, 45, 1233-1246 – Y. Rochat, F. Semet 1994

[Russell 95] Artificial Intelligence: A Modern Approach – Prentice Hall, Englewood Cliffs, NJ. – S. Russell and P. Norvig.1995.

[Smith 04] E-mail correspondence with Peter Smith, Senior Lecturer at City University, regarding "theory-practice gap". – Peter Smith, March 2004

[Solomon 87] Algorithms for the Vehicle Routing and Scheduling Problems with Time Window Constraints – Operations Research, 35 (2):254-265 – M.M. Solomon 1987

[Toth 98] The Granular Tabu Search (and its Application to the Vehicle Routing Problem) – Technical Report, Dipartimento di Elettronica, Informatica e Sistemistica, Universit di Bologna, Italy – P. Toth, D. Vigo 1998 – also in: INFORMS Journal of Computing, vol. 15, no. 4, pp. 333-346 2003

[Tsang 93] Foundations of constraint satisfaction – Academic press – Edward Tsang 1993

[Tuson 00] No Optimisation Without Representation: A Knowledge-Based Systems View of Evolutionary/Neighbourhood Search Optimiser Design – PhD Thesis, Department of Artificial Intelligence, Edinburgh University – A.L. Tuson 2000.

[Wolpert 95] No free lunch theorems for search – Technical Report SFI-TR-95-02-010, Santa Fe Institute – David H. Wolpert and William G. Macready 1995 – also in: IEEE Transactions on Evolutionary Computation, 1, 1997

[Zweben 90] A framework for iterative improvement search algorithms suited for constraint satisfaction problems – Technical Report RIA-90-05-03-1, NASA Ames Research Center, AI Research Branch – M. Zweben 1990 – also in: Proceedings of the AAAI-90 Workshop on ConstraintDirected Reasoning, 1990

[Zweben 94] Scheduling and rescheduling with iterative repair – in Intelligent Scheduling, M. Zweben and M. S. Fox, editors, chapter 8, pages 241--255. Morgan Kaufmann, San Francisco, CA – M. Zweben, B. Daun, and M. Deale 1994