The Bees Algorithms for the Vehicle Routing Problem

Michael J. Dinneen Aisha J.L. Fenton

Department of Computer Science
University of Auckland
Auckland, New Zealand

Abstract

We examine the Pickup and Delivery problem for Full truck load vehicles - an important problem in moving bulk freight. We provide background on the related problems VRP and PDP as well as providing a brief survey of the more common heuristics for these problems. We provide an algorithm based on adapting classic heuristics and provide a set of test instances to benchmark the problem against. We discuss implementation challenges and solutions and present a real-world case study.

Contents

*

Chapter 1

Introduction

1.1 Motivation

1.1.1 Road transport industry

Road transport is a vital part of New Zealand. It is important for providing basic needs: virtually every product grown, made or used in New Zealand will be carried on a truck at least once during its lifetime[17]. And it is important for the New Zealand economy. The success of New Zealand's export industries are inextricably linked to the reliability and cost effectiveness of road transport.

New Zealand is inline with other countries in having 80% of all freight transported by road (by comparison Europe also carries 80% of all freight road). Road transport is particularly important to regional New Zealand and the export industries which drive these local economies. Trucks carry[17]:

- 95% of export fruit
- 86% of export wool
- 85% of export dairy products
- 65% of export logs
- 35% of export meat

Road transport is a also very closely tied to the New Zealand economy. A study by [**?**] showed that a 1% growth in national output requires a 1.5% increase in transport services.

In addition to this Road transport is itself a major contributor to the New Zealand economy. The road transport industry has a total turnover of around $4 billion a year. This equates to

around 1.4% of economic activity nationally and goes as high as 2.5% of economic activity in regional areas such as Southland[17].

Most road transport business runs on tight margins [**?**]. Therefore technology innovations are seen as key area for investment and as having a great potential for enabling the industry to grow. Technology research is focused on providing solutions in the following areas:

- Improving fuel efficiency (e.g. improved engines, improved aerodynamics)
- Reducing environmental impact of Road transport
- Improving utilization of the fleet
- Reducing running costs of a fleet
- Strain on transport network. Limiting factor for economic growth.

Improvements to vehicle routing has benefits in each of these areas. An optimal schedule will reduce the amount of distance travelled to perform the same amount of work, therefore it: reduces fuel costs, reduces environment impact as less work is required to move the same amount of freight, allows more good to be carried for the same cost, allows for more efficient use of the existing road network.

SOMETHING ABOUT LOGGING TRUCKS

1.1.2 Vehicle routing

TODO

Chapter 2

Background

This section provides history and background material on the Vehicle Routing Problem. In particular we review the solution methods that been brought to bear on the Vehicle Routing Problem and some of the classic results reported in the literature.

This chapter is laid out as follows. We start in section 2.1 by reviewing a closely related problem, the Traveling Salesman Problem, which is a cornerstone of the Vehicle Routing Problem and with which it shares many solution approaches. In section 2.2 we informally define what the Vehicle Routing Problem is, with an aim to providing an intuitive understanding of the problem to the reader. We then review in section 2.3 the *Exact Methods* that have been developed to solve the Vehicle Routing Problem. These are distinguished from the other methods we review in that they provide exact solutions, where the globally best answer is produced. We follow this in section 2.4 by reviewing the *Classic Heuristics* that have been developed for the Vehicle Routing Problem. These methods aren't guaranteed to find the globally best answer, but rather aim to produce close to optimal solutions using algorithms with fast running times that are able to scale to large problem instances. In section **??** we review *Meta-heuristic* methods that have been adapted for the Vehicle Routing Problem. These methods provide some of the most competitive results available for solving the Vehicle Routing Problem and are seen as being the state-of-the-art currently. Lastly in section 2.6 we review a modern family of meta-heuristics called *Swarm Intelligence* that has been inspired by the problem solving abilities exhibited by some groups of animals and natural processes. These last methods have become a popular area of research recently and are starting to produce competitive results to many problems. This thesis uses a Swarm Intelligence method for solving the Vehicle Routing Problem.

2.1 TSP introduction and history

The traveling salesman problem can informally be defined as given n points on a map provide a route through each of the n points such that each point is only used once and the total distance travelled is minimized. The problem's name, traveling salesman, comes

from the classic real-world example of the problem: a salesman is sent on a trip to visit n cities. What is order should she visit the cities in, such that she covers the least distance? See figure 2.1

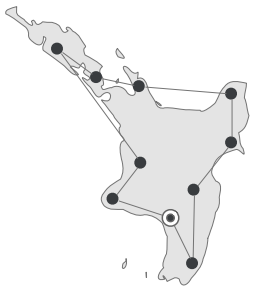

Figure 2.1: Traveling Salesman Problem

TSP is related to a classic graph theory problem, the Hamiltonian path. Hamiltonian circuits have been studied since 1856 by both Hamilton [10] and Kirkman [13]. The traveling salesman problem has been informally discussed probably for many years [22] but didn't become actively studied until after 1928 where Menger, Whitney, Flood and Robinson produced much of the early results in the field. Robinson's RAND report [21] might have been the first article to call the problem by the name it has since become known, the Traveling Salesman Problem.

> The purpose of this note is to give a method for solving a problem related to the traveling salesman problem. One formulation is to find the shortest route for a salesman starting from Washington, visiting all the state capitals and then returning to Washington. More generally, to find the shortest closed curve containing n given points in the plane.

An early result was provided by Dantzig, Fulkerson, and Johnson [5]. Their paper gave an exact method for solving a 49 city problem, a large number of cities for the time. Their algorithm used the cutting plane method to provide an exact solution. This approach has been the inspiration for many subsequent approaches, and is still the bedrock of algorithms that attempt to provide an exact solution.

Karp's famous paper, Reducibility Among Combinatorial Problems, in 1972 showed that the Hamiltonian Circuit problem is NP-complete. This implied the NP-hardness of TSP, and thus supplied the mathematical explanation for the apparent difficulty of finding optimal tours.

A generalization of TSPis MTSP, where multiple routes are allowed to be constructed (i.e. multiple salesman can be used to visit the cities). The pure MTSP can trivially be turned into a TSPby constructing a graph G with $n-1$ additional copies of the starting node to the graph and forbidding travel directly between each n starting nodes. Note however that the pure formulation of MTSP places no additional constraints on a route. In real life applications MTSP is typically employed with additional constraints, such as each route should be of equal size (e.g. work should be balanced among the salesman), or no route should exceed a total distance or time (e.g. a anyone salesman shouldn't be asked to work more than 8 hours a day).

MTSPleads us naturally into the a class of problems derived from the Vehicle Routing Problem (VRP). VRP– and it's family of related problems – can be understood as being a combination of MTSP along with other constraints, which are in themselves often combinatorially hard problems.

2.2 VRP overview

The Vehicle Routing Problem (VRP) seeks to solve the problem of constructing routes for a fleet of vehicles. Each route takes the vehicle through a set of customers to deliver a good (or service) at different locations. The routes must be constructed such that all constraints are meet, classically, the vehicle's capacity isn't exceeded by the number of customers being serviced. The goal is to minimize the cost of distributing the goods, which typically has meant minimizing the distance travelled across all routes.

VRP was first formally defined by Dantzig and Ramser, in [6] and has remained an important problem in logistics and transport (the original name given to the problem presented by Dantzig and Ramser was The Truck Scheduling Problem). The Vehicle Routing Problem is closely related to two problems, the Multiple Traveling Salesman Problem MTSP and the Bin Packing Problem BPP.

2.3 Exact methods

The first efforts were concerned with exact solutions, and proceeded using many of the same techniques brought to bear on TSP. We follow Laporte and Nobert's survey [14] and classify exact algorithms for the VRP into three families: Direct tree search methods, Dynamic programming, and Integer linear programming.

The first classic Direct tree search results are due to Christolds and Ellison. Their 1969 paper in Operations Research Quarterly provided the first branch and bound algorithm for exactly solving the VRP[4]. Unfortunately it's time and memory requirements where such that it was only able to solve problems of up to 13 customers. This result was later improved upon by Christolds in 1976 by using a different branch model. This improvement allowed him to solve for up to 31 customers.

Christofides, Mingozzi, and Toth, [3] provide a lower bound method that is sufficiently quick (in terms of runtime performance) to be used to as a lower bound for excluding nodes from the search tree. They used this method to provide solutions as for a number of problems containing between 15 to 25 customers. Laporte, Mercure and Nobert [**?**] used MTSP as a relaxation of VRP within a branch and bound framework to provide solutions for 'realisticly' sized problems containing up to 250 customers.

The Dynamic Programming approach was first proposed for VRP by Eilon, Watson-Gandy and Christofides (1971). Their approach allowed them to solve exactly for problems of 10 to 25 customers. Since then, Christofides has made improvements to this algorithm to solve exactly for problems up to 50 vertices large.

A Set Partitioning algorithm was given by Balinski, and Quandt in 1964 to produce exact VRP solutions [1]. The problem sets they used where very small however, having only between 5 to 15 customers. And even despite this they weren't able to produce an solution for some of the problems. However, taking their approach as a starting point many authors have been able to produce more powerful methods. Rao and Zionts (1968), Foster and Ryan (1976), Orloff (1976), Desrosiers, Soumis and Desrochers (1984), Agarwal, Mathur and Salkin (1989), and Desrochers, Desrosiers and Solomon (1990), have all extended the basic set partitioning algorithm, using the Column Generation method, to produce more practically useful results.

Notwithstanding the above results, exact methods have been of more use in advancing theoretical understanding of VRP than to providing solutions to real life problems. This can mostly be attributed to the fact that real-life VRP instances often involve hundreds of customers, and involve richer constraints than are modeled in a simple VRP.

2.4 Classic heuristics

Classic VRP heuristics can be classified into three families. Constructive heuristics; Two-phase heuristics, which again can be classified into two sub-families: cluster first and then route, and route first and then cluster; and Improvement methods.

2.4.1 Constructive heuristics

We start by looking at Constructive heuristics. Constructive heuristics build a solutions from the ground up. They typically provide a recipe for building each route, such that the total cost of all routes is minimized. An early and influential result was given by Clarke and Wright in their 1964 paper, Scheduling of vehicles from a central depot to a number of delivery points [8]. In this paper they present a heuristic method for solving VRP problems that improved upon Dantzig and Ramser's work – it is commonly known as the Clarke-Wright heuristic. The heuristic is based on the simple process of combining routes (starting with each route containing a single customer) such that the combination reduces the overall

cost (typically distance) while still producing feasible solutions.

The heuristic has been used to solve problems of up to 1000 customers with results often within 10% of optimal using only a 180 seconds of runtime [24]. This classic algorithm has been extended by Gaskell (1967), Yellow (1970) and Paessens (1988), who have suggested alternatives to the savings formulas used by Clarke and Wright. These approaches typically introduce additional parameters to guide the algorithm towards selecting routes with geometric properties that are likely to produce better combinations.

Alkinkemer and Gavish provide an interesting variation on the basic savings heuristic [12]. They use a matching algorithm to combine multiple routes in a step. To do this they construct a graph where each vertex represents a route, each edge represents a feasible saving, and the edge's weight represents the saving that can be realized by the merge of the two routes. The algorithm proceed by solving a maximum cost weighted matching of the graph.

A group of construction heuristics, know as Sequential Insertion Heuristics, had their first results due to Mole and Jameson [20].

2.4.2 Two-phase heuristics

We next look at two-phase heuristics. We start by looking at the cluster-first, route second sub-family. One of the foundational algorithms for this method is due to Gillett and Miller who provided a new approach called the Sweep Algorithm in their 1974 paper[2]. This popularized the two-phase approach, although this method was suggested earlier by Wren in his 1971 book, and subsequently in Wren and Holliday's 1972 paper for Operations Research Quarterly. In this approach, a initial clustering phase is used to cluster the customers into a base set of routes. From here the routes are treated as separate TSP and optimized accordingly. The two-phase approach typically doesn't prescribe a method for how the TSP is solved and assumes that already developed TSP methods can be used. The classic sweep algorithm uses a simple geometric method to cluster the customers. Routes are built by sweeping a ray, centered at the depot, clockwise around the space enclosing the problem's locations. The Sweep method is surprising effective and has been shown to solve several benchmark VRP problems to within 2% to 9% of the best known solutions [24].

== PIC OF SWEEP ==

Fisher and Jaikumas's 1981 paper builds upon the two-phase approach by providing a more sophisticated clustering method. They solve a General Assignment Problem to form the clusters instead. A limitation of their method is that the amount of vehicle routes must be fixed up front. Their method often produces results that are 1% to 2% better than similar results produced by the classic sweep algorithm [24].

Christofides, Mingozzi, and Toth expanded upon this approach in [**?**] and proposed a method that uses a truncated branch and bound technique (similar to Christofiedes Exact method).

At each step it builds a collection of feasible routes containing a customer i for evaluation. It then evaluates each route and selects the route that the TSPwith the shortest distance can be produced from.

The Petal algorithm is a natural extension to the Sweep Algorithm. It was first proposed by Balinski and Quandt [**?**] and then extended by Foster and Ryan [**?**]. The basic process is to produce a collection of overlapping candidate routes (called petals) and then to solve a set partition problem to produce a feasible solution. As with other two-phase approaches it's assumed that the order of the customers within each routes is solved using any pick of existing TSPheuristics. It has produced some competitive results for small solutions, but quickly becomes impractical where the set of candidate routes that must be considered is large.

Lastly, there are route first, cluster second methods. The basic premise of these techniques are to first construct a "grand" TSP tour such that all customers are visited. The second phase is then concerned with splitting this tour into feasible routes. Route first, cluster second methods are generally thought to be less competitive than other methods [9], although interestingly Haimovich and Rinnooy Kan have shown that if all customers have unit demand then a simple shortest path algorithm (which can be solved in polynomial time) can be used to produce a solution from a TSP tour that is asymptotically optimal [16].

2.4.3 Iterative Improvement Heuristics

Iterative Improvement methods follow a basic hilling climbing approach. They start with a solution, which may have been randomly generated, or produced by another heuristic technique, and then iteratively apply an operation to that solution to improve it. This continues until a local minimum is reached.

There have been a number of operations suggested. Christofides and Eilon gave one of the earliest iterative improvement methods in their paper [4]. In this they made a simple change to the classic TSP operation, 2-opt, increasing the amount of edges removed to three - the operation fittingly being called 3-opt. They found that their heuristics produced better results than the a 2-opt procedure could by itself.

In general operations, such as 3-opt, that remove edges and then search for a more optimal recombination of components take $O(n^y)$ where y is the number of edges removed. A rich strain of research has been on producing operations that reduce the amount of recombinations that must be searched. Or presents an operation that has since come to be known as Or-opt [18]. Or-opt is a restricted 3-opt. It searches for a relocation of all sets of 3 consecutive vertices (i.e. chains), such that an improvement is made. If an improvement can't be made then it tries again with chains of 2 consecutive vertices, and so on. Or-Opt has been shown to produce similar results to that of 3-opt, but with a running time of $O(n^2)$. More recently Renaud, Boctor, and Laptorte have presented a restricted version of 4-opt (in a similar vain to Or-Opt) that has a running time of $O(wn^2)$ [11].

Iterative improvement heuristics are often used in combination with the other techniques. In this case they are run on the candidate solution after the initial heuristics has completed. However, for this purpose there is often a fine balance between producing an operation that improves a solution, and one that is sufficiently destructive enough to escape a local minimum. Recently interest in iterative improvement heuristics has grown because of their use as part of meta-heuristic methods.

2.5 Meta-heuristics

Meta-Heuristics are a broad collection of methods that make few or no assumptions about the type problem being solved. They provide a framework that allows for individual problems to be modeled and 'plugged in' to the meta-heuristic. Typically meta-heuristics take an approach where a candidate solution (or solutions) is initially produced and then is iteratively refined towards the optimal solution. Intuitively meta-heuristics can be thought of searching a problem's problem space. Each iteration searches the neighbourhood of the current candidate solution(s) looking for new candidate solutions that move closer to the goal.

== INSERT PIC OF SEARCH SPACE ==

A limitation of meta-heuristics is that they aren't guaranteed to find an optimal solution (or even a good candidate). Indeed the theoretical underpinnings of what makes a meta-heuristic more effective than another is still poorly understood. Instead meta-heuristics in the literature tend to be tuned for specific problems and then validated empirically.

There have been a number of meta-heuristics produced for the VRP in recent years. Many of the most competitive results produced in the last ten years have been from meta-heuristic approaches. We next review some of the more well known meta-heuristic results for VRP.

2.5.1 Simulated Annealing

Simulated Annealing is inspired by the annealing process used in metallurgy. The algorithm starts with a candidate solution (which can be randomly selected) and then moves to nearby solutions with a probability dependent on the quality of the solution and a global parameter T, which is reduced over the algorithm. In classic implementations the following formula is used to control the probability of a move:

$$e^{-\frac{f(s')-f(s)}{T}}$$

Where $f(s)$ and $f(s')$ represent the solution quality of the current solution, and the new solution respectively. By analogy to the metallurgy process T represents the current temperature of the solution. Initially T is high. This lets the algorithm free itself from any

local minimum that it may be caught in. It is then cooled over time forcing the algorithm to converge to a new solution.

One of the first results was given by Robuste, Daganzo and Souleyrette [7]. They define the search neighbourhood all solutions that can be obtained from the current solution by applying one of three operations: relocating part of a route to another position within the same route, and exchanging customers between two routes. They tested their solution on some large real-world instances of up to 500 customers. They self-reported some success with their method but as their test cases are unique no direct comparison is possible.

Osman has given probably the best know Simulated Annealing results for VRP[19]. His algorithm expands upon many areas of the basic Simulated Annealing approach. The method start by using the Clark and Wright algorithm to produce a starting position. It defines its neighbourhood as candidate solutions that can be searched by applying a $\lambda - interchange$ operation.

$\lambda - interchange$ works by selecting two sequences (i.e. chains) of customers S_p, S_q from two routes, p and q, such that $|S_p|, |S_q| < \lambda$ (note the chains aren't necessary of the same length). The customers within each set are then exchanged until an exchange produces an infeasible solution. As the neighbourhood produced by $\lambda - interchange$ is typically large Osman restricts λ to ≤ 2 and take the first move that provides an improvement.

Osman also uses a sophisticated cooling schedule; his main change being that the temperature is cooled continuously while improvements are found, or if no improvement he resets the temperature (using $T_i = max(\frac{T_r}{2}, T_b)$, where T_r is the reset temperature, and T_b is temperature of the best solution found so far).

Although Simulated Annealing has produced some good results, and in many cases outperforms the classic heuristics (compare [9] with [15]), it is not competitive with the tabu search implementation.

2.5.2 Genetic Algorithms

Genetic Algorithms where first proposed in [**?**]. They've since been applied to many problem domains and are particularly well suited to applications that must work across a few distinct problems. In fact they were the first evolutionary inspired algorithms to be applied to combinatorial problems [**?**]. The basic operation of a GA is as follows:

In classic Genetic Algorithms each candidate solution is encoded as a binary string (i.e. chromosome). Each individual (i.e. candidate solution) is initially created randomly and used to seed the population. There are many techniques suggested in the literature for initially 'bootstraping' the population by making use of other heuristics to produce a stronger initial population, however special care must be taken with Genetic Algorithms to ensure that diversity is maintained within the population - otherwise there is a risk of premature convergence.

Generate the initial population
while *termination condition not meet* **do**
    Evaluate the fitness of each individual
    Select the fittest pairs
    Mate pairs and produce next generation
    Mutate (optional)
end

Next the fittest individuals are selected from the population, and reproduction (crossover) and mutation operations are applyed to them to produce the next generation. A classic crossover operation takes individuals encoded as binary strings and splits them at one or two points. The parts are then recombined to form a new candidate solution. The idea being that the crossover produces a new candidate solution from two successful partial solutions. This process is repeated until a termination condition is meet (often a predetermined running time), or until the population has converged on a fitness.

Special consideration needs to be given to how discrete optimisation problems, such as the VRP, are represented and how their operators are constructed. For instance, consider how the classic crossover operation would work on a TSPpath. When two parts of two separate solutions are combined they are likely to contain duplicates. Therefore it is more common for the VRP (and the TSP as well) to use a direct representation, rather than a binary encoding. In this instance the VRPis represented as collection of sequences, each holdings a sequence of customers. The Genetic Algorithm then must use operators specially designed for this representation of the problem.

Two commonly used crossover operators are Order Crossover (OX) and Edge Assembly Crossover (EAX). OX [**?**] operates by placing two cuts points within each route. The the substring between the two cut points is copied from the first parent directly into the offspring. The remainder of the string from the second parent is then copied to the offspring, but with any duplicates removed. This potentially leaves an incomplete solution, where not all the customers have been routed. The solutions is then "repaired" by inserting any remaining customers using an insertion heuristic.

== PIC OF OX Operator ==

Another common crossover operator is EAX. EAX was originally designed for the TSP but has been adapted to the VRPby [**?**]. EAX operates using the following process:

1. Combine the two candidate solutions into a single graph by merging each solutions edge sets.
2. Create a partition set of the graph's cycles by alternately selecting an edge from each graph.
3. Randomly select a subset of the cycles.
4. Generate a (incomplete) child by taking one of the parents and removing all edges

from the selected subset of cycles. Then add back in the edges from the parent that wasn't chosen.

5. Not all cycles in the child are connected to the route. Repair them by iteratively merging the disconnected cycles to the connected cycles.

== PIC OF EAX ==

An alternative, and interesting, approach found in the literature is to instead encode a set of operations and parameters that are feed to another heuristic, that in turn produces a candidate solution. An early example of this was suggested by [**?**] who encoded a ordering of the customers. The ordering is then feed into an Insertion heuristic to produce the actual candidate solutions.

An influential adaption of Genetic Algorithms for solving VRPTW was given in [**?**] with their GIDEON algorithm. GIDEON uses an approach inspired by the sweep heuristic (an overview is provided in section 2.4.2). Routes are built by sweeping a ray, centered at the depot, clockwise around the space enclosing the problem's locations. Customers are collected into candidate routes based on a set of parameters that are refined by the GA. GIDEON uses the GA to evolve these parameters rather than to operate on the problem directly. Finally GIDEON uses a local search method to optimize customers with the routes, making use of the $\lambda - interchange$ operation (see section 2.5.3 for a description of this operator).

Generally speaking Genetic Algorithms haven't been as competitive in VRP as other meta-heuristics. However, more recently there have been two very promising application of Genetic Algorithms to VRP. Nagata [**?**] has adapted the EAX operator for with the VRP. And Berger and Barkaoui have presented a Hybrid Genetic Algorithm called HGA-VRP in [**?**]. HGA-VRP adapts a construction heuristic to be used as a crossover operator. The basic idea is that a set of routes are selected from each parent that are located close to each other. Customers are then removed from one parent and inserted into the second using an operation inspired by Solomon's construction heuristic for VRPTW[23].

Both methods have reached the best known solution for a number of the classic VRP benchmark instances by Christofides, Mingozzi and Toth [3] and are competitive with the best Tabu Search methods.

2.5.3 Tabu Search

Tabu Search follows the general approach shared by many meta-heuristics; it iteratively improving a candidate solution by searching for improvements within the current solution's neighborhood. Tabu search starts with a candidate solution, that may be generated randomly or by using another heuristic. Unlike Simulated Annealing, the best improvement within the current neighbourhood is always taken as the next move. This introduces the problem of cycling between candidate solutions. To overcome this Tabu Search introduces a

list of solutions that have already been investigated and are forbidden as next moves (hence its name of Tabu List).

The first instance of Tabu Search being used for VRP is by Willard [25]. Willard's approach made use of the fact that VRP instances can be transformed into a MTSP instances and solved. The algorithm makes use of a combination of simple vertex exchange and relocate operations. Although opening the door for further research its results weren't competitive with the best classic heuristics.

Osman gives a more competitive use of Tabu Search in [19]. As with his Simulated Annealing method he makes use of the $\lambda - interchange$ operation to define the search neighbourhood. Osman provides two alternative methods to control how much of the neighbourhood is searched to select the next move - Best-Improvement (BI) and First-Improvement (FI). Best-Improvement searches the entire neighbourhood and selects the move that is the most optimal. First-Improvement searches only until a move is found that is more optimal than the current position. This heuristic produces some competitive results that often out perform the classic heuristics. However it has been refined and improved upon by newer Tabu Search methods.

Toth and Vigo introduced the concept of Granular Tabu Search (GTS) [**?**]. Their method makes use of a process that removes moves from the neighbourhood that are unlikely to produce good results. They reintroduce these moves back into the process if the algorithm is stuck in a local minimum. Their idea follows from an existing idea known as candidate lists. Toth and Vigo's method has produced many competitive results.

Taillard gave a very successful application of Tabu Search in [**?**]. Talliard's Tabu Search uses Or's $\lambda - interchange$ as it's neighbourhood structure. It borrows two novel concepts from [**?**], the use of a more sophisticated tabu mechanism - the duration (or number of iterations) that an item is tabu for is chosen randomly, and a diversification strategy, where vertices that are frequently moved without giving an improvement are penalised. An novel aspect of Taillard's algorithm is it's decomposing of the problem into sub-problems. The problem is split into regions using a simple segmentation of the region centred about the depot (Taillard also provides an alternative approach for problems where the customers are evenly distributed around the depot). From here each subproblem is solved individually, with customers being exchanged between neighboring segments periodically. Taillard observes that exchanging customer beyond neighboring segments is unlikely to produce an improvement, so can safely be ignored. Taillard's method has produced some of the currently best known results for the standard Christofides, Mingozzi and Toth problem sets [3].

2.5.4 Large Neighbourhood Search

Large Neighbourhood Search (commonly abbreviated to LNS) was recently proposed as a heuristic by Shaw [**?**]. Large Neighbourhood Search is a type heuristic belonging to

the family known as Very Large Scale Neighbourhood search (VLSN)[1]. Very Large Scale Neighborhood search is based on a simple premise that rather than searching within a neighborhood of solutions that can be obtained from a single (and typically quite granular) operation, such as $2-opt$, it might be profitable to consider a much broader neighborhood; A neighbourhood of candidate solutions that are obtained from applying many simultaneous changes to the current solution. What distinguishes these heuristics from others is that the neighborhood under consideration is typically exponentially large, and sometimes infeasible to search. Therefore much attention is given to providing practical methods to search these neighborhoods.

Large Neighbourhood Search uses a Destroy and Repair metaphor for how it searches within its neighbourhood. The basic operation is as follows.

---

x = an initial solution
while *termination condition not meet* **do**
  $x^t = x$
  $destroy(x^t)$
  $repair(x^t)$
  if x_t *better than current solution* **then**
    $x = x_t$
  end
end
Result: x

Firstly a starting position is generated. This can be done randomly or by using another heuristic. Then for each iteration of the algorithm a new position is generated by destroying part of the candidate solution and then by repairing it. If the new solution is better than the current solution, then this is selected as the new position and loop repeats. This can be seen as being a type of Very Large Scale Neighborhood because at each iteration the number of neighboring solutions that can be built from the partially destroyed solution is exponential on the size of the items removed (i.e. destroyed).

Obviously a key component of this approach are the functions used to destroy and repair the solution. Care must be given to how these functions are constructed. They must pinpoint an improving solution from a very large neighbourhood of candidates, while also providing enough degrees of freedom to escape local minimum.

Empiric evidence in the literature shows that even surprisingly simple functions can be effective (more effective in some cases) [**?**][**?**]. In applications of Large Neighbourhood Search to VRP a pair of simple operations are commonly used (alongside more complex ones) for the destroy and repair functions. Specifically, part of the candidate solution is destroyed by randomly selecting and removing n customers. Then it is repaired by finding the least cost reinsertion points back into the solution for the n customers.

Shaw applied Large Neighbourhood Search to VRP in his original paper introducing the method[**?**]. Shaw introduces a few novel approaches to the destroy and repair functions.

[1]LNS is somewhat confusingly named given that it a type of VLSN, and not a competing approach

His destroy function removes a set of 'related' customers. He defines related customer to be any two customers that share a similar geographic location, that are sequentially routed, or that share a number of similar constraints (such as overlapping time windows if times constraints are used). The idea of removing related customers, over simply removing random customers, is that related customers are more likely to be profitable exchanged (likewise unrelated customers are more likely to be reinserted back in their original positions). Shaw's repair function made use of a simply branch and bound method that finds the minimum cost reinsertion points within the partial solution. His results where immediately impressive and reached the many of the best known solutions on the Christofides, Mingozzi and Toth problems [3].

More recently Ropke proposed an extension to the basic Large Neighbourhood Search process in [**?**]. His method adds the concept of using a collection of destroy and repairs functions, rather than using a single pair. Which function to use is selected at each iteration based on it's previous performance. In this way the algorithm self adapts to using the most effective function to search the neighbourhood.

Ropke makes use of several destroy functions. He uses a simple random removal heuristic, Shaw's removal heuristic, and a worst removal heuristic, which removes the most costly customers (in terms of that customer's contribution the routes overall cost). Likewise for insertion he makes use of several different functions. These include a simple greedy insertion heuristic, and a novel insertion method he calls the 'regret heuristic'. Informally the regret heuristic reinserts those customers first who are most impacted (in terms of increased cost) by not being inserted into their minimum positions. Specifically let U be the set of customers to be reinserted. let $x_{ik} = \{1, ..., m\}$ be a variable that gives the k'th lowest cost for inserting customer $i \in U$ into the partial solution. Now let $c_i^* = x_{i2} - x_{i1}$, in other words the cost difference between inserting customer i into it's second best position and its first. Now in each iteration of the repair function choose a customer that maximizes:

$$\max_{i \in U} c_i^*$$

Ropke presents a series of results that show that his Large Neighbourhood Search is very competitive for solving the VRP and a large number of related problems, VRP, VRPTW, PDPTW, and DARP). Considering that Large Neighbourhood Search was only proposed in 1998 it has been very successful. In a short space of time it has attracted a large amount of research and has produced some of the most competitive results.

2.6 Swarm intelligence

A recent area of research is in producing heuristics that mimic certain aspects of Swarm behaviour. Probably the most well known heuristics in this class are Particle Swarm Optimisation (PSO) and Ant Colony Optimisation (ACO). Swarm Intelligence is interesting to combinatorial optimisation researches as it demonstrates a form of emergent intelligence, where individual members with limited reasoning capability and simple behaviours, are able to achieve collective goals that are remarkable sophisticated.

In the context of combinatorial optimisation these behaviours can be mimicked and exploited to produce algorithms that are able to produce solutions to complex problems by simulating a number of agents who in themselves only need to perform rudimentary operations. A feature of this class of algorithms is the ease with which they can be parallelized, making them more easily adaptable to large scale problems.

Swarm Intelligence algorithms have been employed to solve a number of problems. We look at two examples here, Ant Colony Optimisation, and the Bees Algorithm, which this thesis makes use of.

2.6.1 Ant Colony Optimization

Ant Colony Optimization is inspired by how ants forage for food and communicate successful sites back to the colony. Real life initially forage for food randomly. Once they find a food source they return to the colony and in the process lay down a pheromone trail. Other ants that then stumble upon the pheromone trail follow it with a probability dependent on how strong (and old) the pheromone trail is. If they do follow it and find food, then they return too, strengthening the pheromone trail. The strength of the pheromone trail reduces over time meaning that younger and shorter pheromone trails, that do not take as long to traverse, attract more ants.

== PIC OF ACO ON GRAPH ==

Ant Colony Optimisation mimics this behaviour on a graph by simulating ants marching along a graph that represents the problem being solved. The basic operation of the algorithm is as follows:

Data: A graph representing the problem
while *termination condition not meet* **do**
    positionAnts()
    while *solution being built* **do**
        marchAnts()
    end
    updatePheromones()
end

At each iteration of the algorithm the ants are positioned randomly within the graph. The ants are then stochastically marched through the graph until they have completed a candidate solution (in the case of a TSPthis would be a tour of all vertices). At each stage of the march each ant selects their next edge based on the following probability formula:

$$p_{ij}^{k} = \frac{[\tau_{ij}^{\alpha}][\eta_{ij}^{\beta}]}{\sum_{l \in N^{k}}[\tau_{il}^{\alpha}][\eta_{il}^{\beta}]}$$

Where p_{ij}^k is probability that ant k will traverse edge ij, N^k is the set of all edges that haven't been traversed by ant k yet, τ is the amount of pheromone that has been deposited at an edge, η is the desirability of an edge (based on a priori knowledge specific to the problem), and α and β are global parameters that control how much influence each term has.

Once the march is complete and a set of candidate solutions have been constructed (by each ant k), update the pheromones deposited on each edge by the follow equation:

$$\tau_{ij} = (1 - \rho)\tau_{ij} + \sum_{k=1}^{m} \Delta\tau_{ij}^k$$

Where $0 < \rho \leq 1$ is the pheromone persistence, and $\Delta\tau_{ij}^k$ is a function that gives the amount of pheromone deposited by ant k. The function is defined as:

$$\Delta\tau_{ij}^k = \begin{cases} 1/C^k & \text{if edge } ij \text{ is visited by ant } k \\ 0 & \text{otherwise} \end{cases}$$

Where C^k represents the total distance traveled through the graph by ant k. This ensures that shorter paths result in more pheromone being deposited.

As an example of Ant Colony Optimization's use in combinatorial problems, we show how it can be applied to the TSP. We build a weighted graph with $i \in V$ representing each customer and $ij \in E$ and w_{ij} representing the cost of travel between each customer. And at each stage of the iteration we ensure that the following constraints are met:

- Each customer is visited at most once
- We set η_{ij} to be equal to w_{ij}

When the Ant Colony Optimizer starts it positions each ant at random customer within the graph. Each step of the ant's march then builds a tour through the customers. Once a ant has completed a tour then we have produced a candidate solution for the TSP. The solutions will be of variable quality. We use the length of the tours to ensure that more pheromone is deposited on shorter tours. At the end of n iterations the ants will have converged on an optimal solution (but like all meta-heuristics there's no guarantee that this the global optimum).

== PIC OF IT BEING USED ON A TSP ==

Ant Colony Optimization has been applied to VRP by Bullnheimer, Hartl, and Strauss in [**?**], [**?**]. They adapted the straight implementation used for the TSP (detailed immediately above) by forcing the ant to create a new route each time it exceeds the capacity or maximum

distance constraint. They also use a modified edge selection rule that also takes into account the vehicle's capacity and it's proximity to the depot. The updated rule is now given by:

$$p_{ij}^{k} = \frac{[\tau_{ij}^{\alpha}][\eta_{ij}^{\beta}][s_{ij}][\kappa_{ij}]}{\sum_{l \in N^k}[\tau_{il}^{\alpha}][\eta_{il}^{\beta}][s_{il}][\kappa_{il}]}$$

Where s represents the proximity of customers i, j to the depot, and $\kappa = (Q^i + q^j)/Q$ (Q giving the max capacity, Q^i giving the capacity already used on the vehicle, and q^j is the additional load to be added). κ influences the ants take advantage of available vehicle capacity.

Bullnheimer et al.'s implementations of Ant Colony Optimization for VRP produces good quality solutions for the Christofides, Mingozzi and Toth problems [3], but is not competitive with the best modern meta-heuristics.

More recently Reimann, Stummer, and Doerner have presented a more competitive implementation of Ant Colony Optimization for VRP [**?**]. Their implementation operates on a graph where $ij \in E$ representing the savings of combining two routes, as given by the classic Clark and Wright savings heuristic [8]. Each ant selects an ordering of how the merges are applied. This implementation is reported to be competitive with the best meta-heuristics [**?**].

2.6.2 Bees Algorithm

Over the last decade, and inspired by the success of Ant Colony Optimisation, there have been a numerous algorithms proposed that aim to mimic the behaviour of bees. These includes Bee colony optimization that has been applied to many different combinatorial problems, Marriage in honey Bees Optimization (MBO) that has been applied to propositional satisfiability problems, BeeHive that has been applied to timetabling problems, the Virtual Bee Algorithm (VBA) that is used for function optimisation, Honey-bee mating optimisation (HBMO) for cluster analysis, and finally the Bees Algorithm that is the focus of this thesis. See [**?**] for a bibliography and high level overview on many of these algorithms.

=== CAN EXTEND OUT IF NEEDED ===

The Bees Algorithm was first proposed by [**?**]. It is inspired by the foraging behaviour of honeys bees. Bee colonies must search a large geographic area around their hive in order to find sites with enough pollen that it can sustain a hive. Its essential that the colony makes the right choices in what sites are exploited and how much resource is expended on a particular site. They achieve this by sending scout bees out in all directions from the hive. Once a scout bee has found a promising site it returns to the hive and recruits hive mates to forage at the site too. The bee does this by performing a *waggle* dance. The dance communicates the location and quality of the site (i.e. fitness). Over time as more bees successfully forage at the site, more are recruited into exploiting that site. This last aspect

is similar in process to how ants divert more resource to promising areas when foraging.

== PIC OF WAGGLE DANCE ==

Informally the algorithm can be described as follows. Bees are initially sent out to random locations. The fitness of each site is then calculated. A proportion of the bees are reassigned to those sites that had the highest fitness values. Here each bee searches the local neighbor of the site looking to improve site's fitness. The remainder of the bees are sent out scouting for new sites, in other words they are reinitialized to a random position. This process repeats until some site reaches a satisfactory level of fitness – or the optimal solution is reached, if this is known in advance.

More formally the algorithm operates as follows:

Algorithm 1: IntervalRestriction

$B = \{b_1, b_2, ..., b_n\}$ setToRandomPosition(B)
while *termination condition not meet* **do**
    sortByFitness(B)
    $E = \{b_1, b_2, ..., b_e\}$
    $R = \{b_{e+1}, b_{e+2}, ..., b_m\}$
    searchNeighbourhood($E \cup \{c_1, ..., c_{nep}\}$)
    searchNeighbourhood($R \cup \{d_1, ..., d_{nsp}\}$)
    setToRandomPosition($B - (E \cup R)$)
end

B is the set of bees that are used to search the search space. Initially the bees are set to random positions. Function $sortByFitness$ sorts the bees in order of maximum fitness. It then proceeds by taking the m most promising sites found by the bees. It does this by partitioning these into two sets, $E, N \subset B$. E is the first e best sites, and represents the so called *elite* bees. N is the $m - e$ next most promising sites. The $searchNeighbourhood$ function explores the neighbourhood around a provided set of bees. Each sites in E and N is explored. nep bees are recruited for the search of each $b \in E$, and nsp are recruited for the search of each $b \in N$. In practice this means that nep and nsp moves are explored, respectively, within the neighbourhoods of each site. These moves are typically made stochastically, but it is imaginable that a deterministic approach could be used too. The remaining $n - m$ bees (in other words, those not in E and N) are set to random positions. This is repeated until the termination conditions is met, which may be a running time threshold or a predetermined fitness level.

The promised advantage of the Bees Algorithm over other meta-heuristics is its ability to escape local minima and its ability to navigate search topologies with rough terrain (such as in figure 2.2). It achieves this by scouting the search space for the most promising sites, and then by committing more resource to those sites that are producing results.

The Bees Algorithm has been applied to manufacturing cell formation, training neural networks for pattern recognition, scheduling jobs for a production machine, data clustering,

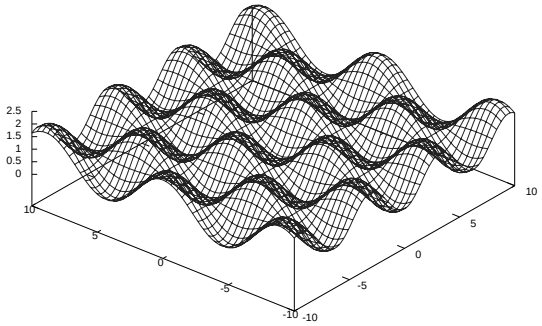

Figure 2.2: Search space with a rough terrain

and many others. See [**?**] for more examples and a comprehensive bibliography. However, as best as we know the Bees Algorithm hasn't been used on the Vehicle Routing Problem until now.

Chapter 3

Problem Definition

- Full-truck-load pickup and deliveries
- Multiple trucks
- Start at depot. Return to depot
- Time window constraints

3.1 Industry

Full truck load (FTL) movement of freight is a common. Most bulk materials are moved as full truck loads. The classic VRP problem examines the case where a truck is Less than Truck Load (LTL). In this case a truck is loaded with many goods that are to be transported to many different customers. The truck is contained by its capacity and sometime also by the length of the trip. The VRP maps well to the real-world freight business of distribution where once goods have been moved between major centres then they are distributed out to retailers or end customers.

Although in the literature VRP has been extensively studied a comparatively small amount has been published on PDP. As far as we know no papers have been published on the PDP for full truck loads.

In transporting bulk materials, such as Logging, Petroleum, Cement, ? the more common model is where the truck is filled to capacity at the primary manufacturing plant (e.g. cement plant, forestry skid site) and then distributed as a FTL (or many FTL) to a resellers such as a concrete plant or saw mill. In bulk transport the transport planner will try to minimize the amount of empty running by maximizing the amount of back-loading possible.

Empty-running is where a vehicle is traveling between jobs without any goods. For the bulk transport planner this total distance traveled isn't important as their rates are structured

so that this travel is reflected in the cost of the job. A bulk transport company on the other hand isn't paid for time spent traveling between jobs.

A back-load is simply were a series of pickups and deliveries can be chained together such that empty-running time is minimized.

3.1.1 Operation

3.1.2 Objectives

- Minimize between job travel distance

3.1.3 Constraints

3.2 Formal definitions

3.2.1 Classic VRP

We formulate the VRP here as an integer programming problem. Although it is possible to solve the VRP as an integer programming problem this approach is only feasible for small problem sizes. Rather the problem is presented here in this fashion so that it can be stated precisely.

The VRP is defined on a graph (V, E). The vertices of the graph V represent customers with v_0 and v_{n+1} representing the depot where the vehicle starts and ends each route. Customers are denoted by $C = 1, 2, ..., n$. The set of edges E corresponds to possible connections between customers. For our use here all connections are possible – that is it is possible for a vehicle to drive between any two customers – except where the depot is concerned. No edge terminates at v_0 and no edge originates at v_{n+1} and there is no edge $0, n + 1$. Each edge $i, j \in E$ has a corresponding cost c_{ij} which typically represents the travel distance between customers.

The set of routes is denoted by R and each route has a maximum capacity q. Each customer has a demand $d_i, i \in C$.

The model contains the decision variable X_{ij}^r which is defined $\forall i, j \in E, \forall r \in R$. X_{ij}^r is equal to 1 if route r contains a trip from customer c_i to customer c_j and 0 otherwise.

(3.1)

Minimize:

$$\sum_{r \in R} \sum_{ij \in E} c_{ij} X_{ij}^r \tag{1}$$

Subject to:

$$\sum_{r \in R} \sum_{j \in V} X_{ij}^r = 1 \qquad \forall i \in C \tag{2}$$

$$\sum_{i \in C} d_i \sum_{j \in C} X_{ij}^r \leq q \qquad \forall r \in R \tag{3}$$

$$\sum_{j \in V} X_{0j}^r = 1 \qquad \forall r \in R \tag{4}$$

$$\sum_{j \in V} X_{j0}^r = 1 \qquad \forall r \in R \tag{5}$$

$$\sum_{i \in V} X_{ik}^r - \sum_{j \in V} X_{kj}^r = 0 \qquad \forall k \in C \text{ and } \forall r \in R \tag{6}$$

The objective function (1) aims to minimize costs c_{ij}. Constraint (2) states that each customer can only be serviced by a single route. Constraint (4) enforces the capacity constraint: each route can not exceed the maximum vehicle capacity. Constraint (5) and (6) ensure that each route starts at exactly 1 depot vertex and finishes at exactly one depot vertex. Lastly, constraint (7) is a flow constraint that ensures that the number of vehicles entering a customer is equal to the number of vehicles leaving.

3.2.2 PDP

We now similarly define the PDP.The PDP generalizes the VRP. Goods can now be picked up from a customer or delivered to a customer along a single route. In contrast in the VRP all jobs are either picking up goods or delivery goods to a customer. The PDP is defined on a graph (V, E). The vertices of the graph V represent all pickup and delivery locations along with v_0 and v_{2n+1} representing the depot where the vehicle starts and ends each route. Pickup jobs are denoted by $P = \{p_1, p_2, ..., p_n\}$ and delivery jobs by $D = \{d_1, d_2, ..., d_n\}$. The set of edges E corresponds to the possible connections between jobs. For our use here all connections are possible – that is it is possible for a vehicle to drive between any two jobs – except where the depot is concerned. No edge terminates at v_0 and no edge originates at v_{2n+1} and there is no edge $0, 2n + 1$. Each edge $i, j \in E$ has a corresponding cost c_{ij} which typically represents the travel distance between customers.

The set of routes is denoted by R and each route has a maximum capacity q. Each location has a demand $d_i \in V$. Pickup jobs have a positive demand whereas deliveries are assumed to have a negative demand. A route containing a pickup $p_i \in P$ must also contain its corresponding delivery $d_i \in D$.

The model contains the two decision variables: X_{ij}^r which is defined $\forall i, j \in E, \forall r \in R$. X_{ij}^r is equal to 1 if route r contains a trip from location v_i to location v_j and 0 otherwise.

Let y_i denote a an intermediate variable that stores the total load of a vehicle traveling route r and visiting location v_i.

(3.2)

Minimize:

$$\sum_{r \in R} \sum_{ij \in E} c_{ij} X_{ij}^r \tag{1}$$

Subject to:

$$\sum_{r \in R} \sum_{j \in V} X_{ij}^r = 1 \qquad \forall i \in V \tag{2}$$

$$\sum_{j \in V} X_{0j}^r = 1 \qquad \forall r \in R \tag{4}$$

$$\sum_{j \in V} X_{j0}^r = 1 \qquad \forall r \in R \tag{5}$$

$$\sum_{i \in V} X_{ik}^r - \sum_{j \in V} X_{kj}^r = 0 \qquad \forall k \in C \text{ and } \forall r \in R \tag{6}$$

$$p_i \in r \Rightarrow d_i \in r \forall r \in R \tag{7}$$

$$p_i \text{ appears before } d_i \in r \forall r \in R \tag{8}$$

$$X_{ij}^r = 1 \Rightarrow y_i + d_i = y_j \forall r \in R \text{ and } \forall i, j \in V \tag{9}$$

$$y_0 = 0 \tag{10}$$

$$\sum_{r \in R} \sum_{j \in V} X_{ij}^r y_i \leq q \qquad \forall i \in V \tag{11}$$

The objective function (1) aims to minimize costs c_{ij}. Constraint (2) states that each customer can only be serviced by a single route. Constraint (4) and (5) ensure that each route starts at exactly 1 depot vertex and finishes at exactly one depot vertex. Constraint (6) is a flow constraint that ensures that the number of vehicles entering a customer is equal to the number of vehicles leaving.

Constraint (7) states that jobs pickup and delivery jobs are serviced by the same route. Constraint (8) enforces that goods are picked up before being delivered. And constraints (9) through (11) ensure that each vehicle's capacity isn't exceeded.

3.2.3 PDP-FTL

The mathematical model can be represented by PDP above. As the demand on each job fills the truck we can simplify the model so that each pickup and delivery constraint is paired. We also add two additional constraints: vehicles fleet size is fixed and each route has a maximum distance travelled (this is used to limited the work undertaken to a shift). These aren't required for the general case but are important in most real-world applications of the problem.

We start by using the same framework given in the PDP formulation. We replace the PDP concept of pickup and delivery locations, with the simpler notation of jobs $J = \{1, 2, ..., n\}$. Each job $j\ inJ$ is represented by the arc j_p, j_d which represents the pickup and delivery locations of the job. Each job can only be serviced by a single route $rinR$.

The number of routes is set to the fixed number of vehicles we have available $|R| = k$.

(3.3)

Minimize:

$$\sum_{r \in R} \sum_{ij \in E} c_{ij} X_{ij}^r \tag{1}$$

Subject to:

$$\sum_{r \in R} \sum_{j \in V} X_{ij}^r = 1 \qquad \forall i \in V \tag{2}$$

$$\sum_{j \in V} X_{0j}^r = 1 \qquad \forall r \in R \tag{4}$$

$$\sum_{j \in V} X_{j0}^r = 1 \qquad \forall r \in R \tag{5}$$

$$\sum_{i \in V} X_{ik}^r - \sum_{j \in V} X_{kj}^r = 0 \qquad \forall k \in C \text{ and } \forall r \in R \tag{6}$$

$$p_i \in r \Rightarrow d_i \in r \forall r \in R \tag{7}$$

$$p_i \text{ appears before } d_i \in r \forall r \in R \tag{8}$$

$$X_{ij}^r = 1 \Rightarrow y_i + d_i = y_j \forall r \in R \text{ and } \forall i, j \in V \tag{9}$$

$$y_0 = 0 \tag{10}$$

$$\sum_{r \in R} \sum_{j \in V} X_{ij}^r y_i \leq q \qquad \forall i \in V \tag{11}$$

The objective function (1) aims to minimize costs c_{ij}. Constraint (2) states that each job can only be serviced by a single route. Constraint (4) enforces the capacity constraint: each

route can not exceed the maximum vehicle capacity. Constraint (5) and (6) ensure that each route starts at exactly 1 depot vertex and finishes at exactly one depot vertex. Constraint (7) is a flow constraint that ensures that the number of vehicles entering a customer is equal to the number of vehicles leaving.

Constraint (8) states that jobs pickup and delivery jobs are serviced by the same route. Constraint (9) enforces that goods are picked up before being delivered. And constraints (10) through (**??**) ensure that each vehicle's capacity isn't exceeded.

Objectives

- Minimize between job travel distance

Constraints

3.2.4 VRPTW

3.2.5 PDPTW

IN REAL WORLD CASE OFTEN VEHICLE NUMBER IS SET. IN THIS CASE WE WANT TO MAKE MAXIMUM USE OF THE VEHICLES THAT ARE AVAILABLE SINCE WAGES ARE PAID REGARDLESS.

Chapter 4

Algorithm

4.1 Goals

Design goals:

- Fast
- Minimal
- Take advantage of modern parallel hardware

4.2 Problem Representation

4.3 Modified Bees algorithm

4.3.1 Deeper driller

4.3.2 Keeping Bees separate

4.3.3 Tabu List

4.3.4 Aged sites

4.3.5 Selection of when to wonder around

4.4 Problem Initiation

4.5 Large Neighbourhood Search

4.5.1 Destroy

4.5.2 Repair

Chapter 5

Results

5.1 Test instances

5.2 Results

5.3 Comments

Chapter 6

Conclusion

6.1 Future Directions

Chapter 7

Appendix A - Implementation

This chapter details how the optimization system is implemented. The system's architecture, technologies employed in the system and optimisations done to the system are detailed here.

7.1 Parallelization

Actor Model

7.2 Source Code

Get it here ...

Bibliography

[1] M.L. Balinski and R.E. Quandt. On an integer program for a delivery problem. *Operations Research*, 12(2):300–304, 1964.

[2] L.R. Miller B.E. Gillett. A heuristic algorithm for the vehicle dispatch problem. *Operations Research*, 22:340–349, 1974.

[3] Mingozzi A. Christofides, N. and P. Toth. Exact algorithms for the vehicle routing problem, based on spanning tree and shortest path relaxations. *Mathematical Programming*, 20:255–282, 1981a.

[4] N. Christofides and S. Eilon. An algorithm for the vehicle dispatching problem. *ORQ*, 20:309–318, 1969.

[5] Fulkerson Dantzig and Johnson. Solution of a large-scale traveling salesman problem. *Operations Research*, (2):393410, 1954.

[6] G. B. Dantzig and J. H. Ramser. The truck dispatching problem. *Management Science*, 6(1):80–91, Oct, 1959.

[7] C.F. Daganzo F. Robuste and R. Souleyrette II. Implementing vehicle routing models. *Transportation Research*, 24B:263–286, 1990.

[8] J.W. Wright G. Clark. Scheduling of vehicles from a central depot to a number of delivery points. *Operations Research*, 12:568–581, 1964.

[9] Frederic Semet Gilbert Laporte. Classical heuristics for the vehicle routing problem. Technical report, Les Cahiers du Gerad, 1999.

[10] W.R. Hamilton. Memorandum respecting a new system of roots of unity (the icosian calculus). *Philosophical Magazine*, 12, 1856.

[11] G Laporte J Renaud, F F Boctor. A fast composite heuristic for the symmetric traveling salesman problem. *INFORMS Journal on Computing*, 8:134–143, 1996.

[12] B. Gavish K. Altinkemer. Parallel savings based heuristic for the delivery problem. *Operations Research*, 39:456–469, 1991.

[13] T.P. Kirkman. On the representation of polyhedra. *Philosophical Transactions of the Royal Society of London Series A*, 146:413–418, 1856.

[14] G. Laporte and Y Nobert. *Surveys in Combinatorial Optimization*, chapter Exact algorithms for the vehicle routing problem. 1987.

[15] J Potvin M Gendreau, G Laporte. Metaheuristics for the vehicle routing problem. Technical report, Les Cahiers du Gerad, 1998, revised 1999.

[16] A H G R Kan M Haimovich. Bounds and heuristics for capacitated routing problems. *Mathematics of Operations Research*, 10:527–542, 1985.

[17] Road Transport Forum NZ. Road transport forum nz - transport facts. Website.

[18] I Or. *Traveling salesman-type combinatorial problems and their relation to the logistics of regional blood banking*. PhD thesis, Evanston, IL: Northwestern University, 1976.

[19] I. H. Osman. Metastrategy simulated annealing and tabu search algorithm for the vehicle routing problem. *Annals of Operations Research*, 41:421–451, 1993.

[20] S.R. Jameson R.H. Mole. A sequential route-building algorithm employing a generalized saving criteria. *Operations Research*, 27:503–511, 1976.

[21] J. Robinson. On the hamiltonian game (a traveling salesman problem). *Research Memorandum RM-303*, 1949.

[22] Alexander Schrijver. On the history of combinatorial optimization (till 1960). *?*, ?:?, ?

[23] M. M. Solomon. Algorithms for the vehicle routing and scheduling problems with time window constraints. *Oper. Res.*, 35(2):254–265, 1987.

[24] Paolo Toth and Daniele Vigo, editors. *The vehicle routing problem*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2001.

[25] J. A. G. Willard. Vehicle routing using r-optimal tabu search., 1989.