Applying Monte Carlo Techniques to the Capacitated Vehicle Routing Problem

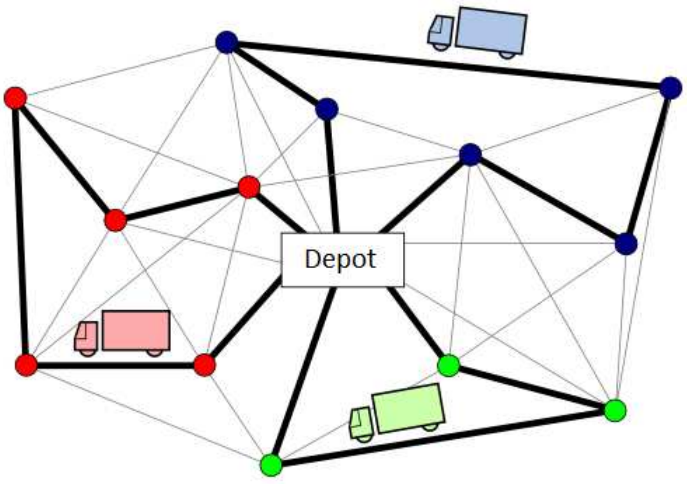
Master Thesis

Frank Takes (ftakes@liacs.nl)

Supervisors:
dr. Walter Kosters (kosters@liacs.nl)
dr. Hendrik Jan Hoogeboom (hoogeboo@liacs.nl)

Master (MSc) Computer Science
Leiden Institute of Advanced Computer Science (LIACS)
Leiden University, The Netherlands

February 26, 2010



Preface

Motivation

This is the Master Thesis of my Computer Science Master studies at Leiden University. I wanted a subject that I found interesting, in line with previous projects, and with a real-life application for my final Master Project, which resulted in this Master Thesis. Before this I have been working on Sokoban and SameGame, two games/puzzles with related NP-complete optimization problems. During these projects I have learned a lot about optimization techniques, solving methods and complexity issues. For my final project I wanted to extend my knowledge about these subjects to a more real-life problem, and while browsing the internet I came across the Vehicle Routing Problem.

Compared to the previous two subjects from Combinatorial Game Theory that I studied, the amount of research done on the Vehicle Routing Problem is huge. I found it an interesting and worthy challenge to study the previous work and extract the relevant pieces for this thesis. After this process, I tried to apply my knowledge from previous projects and studies to the Vehicle Routing Problem, and I wanted to see if I could apply new techniques such as Monte Carlo Simulation to the Vehicle Routing Problem.

During my studies and even more during this project I discovered that Monte Carlo Techniques intrigued me a lot. Specifically, I found it interesting to see how a method as simple as Monte Carlo Simulation could solve or optimize extremely difficult problems that we cannot seem to solve any further by applying traditional techniques. The technique often shows the so-called "emerging behavior", or "emerging intelligence", one might say. There is still a lot to discover on this subject and I expect Monte Carlo Techniques to play a significant if not great role in the development of the whole field of Artificial Intelligence.

As said before, I choose this subject partly because it has a very clear real-life application. Attempting to seek for real-life testing data and algorithms that were doing the vehicle routing job, I contacted multiple national and local package transporting companies. Unfortunately none of them responded positively to my request for an information exchange. Nevertheless I was able to find an excellent test set for my project, so the lack of practical data and information was not a problem at all.

Acknowledgments

This work was done under supervision of dr. Walter Kosters from LIACS, Leiden University. Many thanks to Walter for all the support and interesting talks we had about, amongst others things, this project. Second supervisor for this project was dr. Hendrik Jan Hoogeboom, also from LIACS. I would like to thank Hendrik Jan for his useful comments that really helped improve this thesis.

I would also like to thank Fenne for all her love, patience and care which really helped me to complete my master project successfully. And last but definitely not least I would like to express my gratitude towards my parents, Willem and Lidia, who have always supported me in every possible way during my entire studies.

Contents

Introduction

Vehicle Routing Problem

The Vehicle Routing Problem (VRP) is a widely studied combinatorial optimization problem that was introduced — by now half a century ago — in 1959 by Dantzig and Ramser in [11]. The Vehicle Routing Problem is defined in [41] as the problem of serving a number of customers with a fleet of vehicles, minimizing the cost of distributing the goods. We will see in Chapter 1 that because of the many variants of the Vehicle Routing Problem, this definition is acceptable but actually does not cover the entire problem.

Other than it being a very interesting problem to study for computer scientists, the problem is also important to the industry sector, where it has applications in the fields of transportation, distribution and logistics. Large package shipping companies benefit greatly from implementing the Vehicle Routing Problem as efficiently as possible: every percentage saved on transportation costs means saving tremendous amounts of money.

Monte Carlo Techniques

The term “Monte Carlo Techniques” is used to describe a class of algorithms that relies on some use of random sampling to finally acquire a solution to a given problem. For example in Monte Carlo Simulation, the decisions made when traversing through the search space are based on performing random simulations for the possible successor states. Monte Carlo Techniques are often used when traditional heuristic methods fail, most often because it is hard to derive an admissible evaluation function to determine which of the candidate successors is to be selected next. These techniques are nowadays widely used in all kinds of optimization algorithms, and, as we will see, can also be applied to the Vehicle Routing Problem.

Organization

Chapter 1 starts with a description of the Vehicle Routing Problem and gives an overview of its variants. Several solution methods are presented in Chapter 2. Monte Carlo Techniques are introduced in Chapter 3, and Chapter 4 describes how these techniques can be applied to the Vehicle Routing Problem. We will also propose a new Monte Carlo based algorithm and give an outline of how it performs compared to existing techniques. Chapter 5 concludes the thesis.

Chapter 1

The Vehicle Routing Problem

1.1 Introduction

This first chapter describes the Vehicle Routing Problem in detail, starting with a formal definition of the concepts, various notations, and a description of how the problem is related to the Traveling Salesman Problem. Next, the main variants of the problem are outlined in various subsections. The chapter ends with a section devoted to the complexity of the Vehicle Routing Problem.

1.2 Problem Definition

Let us start with an introduction to the concepts that are used within the **Vehicle Routing Problem (VRP)**. A **customer** is an entity that has a certain **demand** and therefore requires the presence of a **vehicle**, a unit that can move between customers and the **depot**, a unit that initially possesses the demands of the customers. The **fleet** is defined as the total group of vehicles. Moving a vehicle between the depot and the customers comes with a certain **cost**. A **route** is a sequence of visited customers by a certain vehicle, starting and ending at a depot. The **goal** of the Vehicle Routing Problem is to serve all customers, minimizing the total cost of the routes of all vehicles. A visual example is given in Figure 1.1 (to keep visualization simple, the graph in this figure is not complete).

The underlying structure of the VRP is a complete graph $G(V, E)$ with cost matrix C:

- $V = \{v_0, v_1, v_2, \ldots, v_n\}$ is a set of $n + 1$ $(n \geq 1)$ vertices. We distinguish the depot v_0 and exactly n customers $\{v_1, v_2, \ldots, v_n\}$.
- $E = \{(v_i, v_j) \mid 0 \leq i, j \leq n, i \neq j\}$ is the set of $|V| * (|V| - 1)$ (directed) edges (arcs) between the vertices, called the **roads**. If the distance between two vertices is identical in both directions, the restriction $i < j$ is added, and we talk about the **symmetric** variant of the problem.

- $C = (c_{ij})$ is a matrix, where $c_{ij} \geq 0$ is the distance corresponding to edge (v_i, v_j); c_{ii} is always equal to 0. We will also denote $c_{ij} = c(v_i, v_j)$. Depending on whether or not the VRP is symmetric (see Section 1.3), $c_{ij} = c_{ji}$. Also, the triangle inequality is generally assumed to hold here: $c_{ij} \leq c_{ik} + c_{kj}$ $(0 \leq i, j, k \leq n)$.

In essence, an (extremely) simplified variant of the Vehicle Routing Problem is the **Traveling Salesman Problem (TSP)**. The **goal** in the TSP is to find the shortest possible tour (in terms of total distance) that visits each of the cities exactly once. This corresponds to the instance of the VRP where the cost is only dependent on and directly proportional to the distance, there is only one vehicle, and no further (capacity) restrictions apply. In that case we could define a route as a vector $R = (v_0, v_1, \ldots, v_{k+1})$, with $v_0 = v_{k+1}$, $0 \leq k \leq n$. The first restriction, $v_0 = v_{k+1}$, is to ensure that the route starts and ends at the depot. The goal would be to minimize the function $\sum_{j=0}^{k} c_{j,j+1}$.

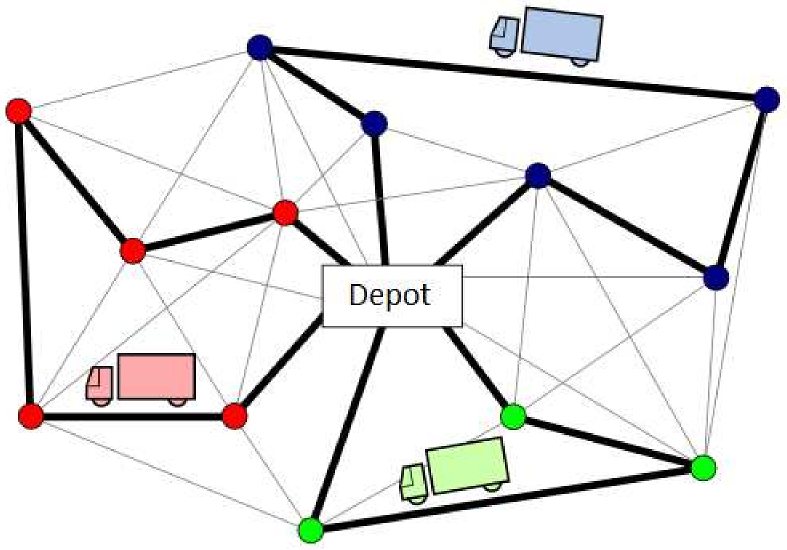


Figure 1.1: A solution for an instance of a VRP with 13 customers and 3 vehicles.

The instance of the Vehicle Routing Problem that accommodates multiple vehicles is the **Multiple Traveling Salesmen Problem (MTSP)**, the variant of the TSP in which there are multiple salesmen walking around. We extend the previous itemized list with the following definitions to work towards a definition of the VRP:

- m, with $m \geq 1$, is defined as the number of vehicles, or the **fleet size**.
- $R_i = (v_0^i, v_1^i, \ldots, v_{k_i}^i, v_{k_i+1}^i)$ is the vector of the route of vehicle i (with $v_0^i = v_{k_i+1}^i = v_0$, $v_j^i \neq v_\ell^i$, $0 \leq j < \ell \leq k_i$), starting and ending at the depot. Here k_i is the length of route R_i.
- $S = \{R_1, R_2, \ldots, R_m\}$ is a set of routes representing the **solution** of a VRP/MTSP instance.
- $C(R_i) = \sum_{j=0}^{k_i} c(v_j^i, v_{j+1}^i)$ is defined as the **cost** of route R_i.

- $C(S) = \sum_{i=1}^{m} C(R_i)$ is defined as the total cost of solution S, satisfying $R_i \cap R_j = \{v_0\}$ for all routes R_i and R_j ($1 \leq i, j \leq m$, $i \neq j$), and $\cup_{i=1}^{m} R_i = V$, so that each customer is served exactly once. Here we treat the route vectors as sets.
- $C(S) \to min$ is the minimization task at hand in the VRP/MTSP.

We have now defined the Vehicle Routing Problem that we introduced in the beginning of this section in terms of a minimization problem. However, there is more to these customer demands than simply vehicles “visiting” the customers. These additional customer’s demands are specified in more detail in the following definitions:

- $d = (d_0, d_1, \ldots, d_n)$ with all $d_i > 0$ ($1 \leq i \leq n$), where n is again the number of customers, is a vector of the customer **demands**; d_0, the demand of the depot, is always equal to 0.
- δ_i is defined as a function for the **service time**: the time to unload all goods at customer v_i ($1 \leq i \leq n$). This function δ is often dependent on the size of the demand of that customer. We will also denote $\delta_i = \delta(v_i)$.
- $C(R_i) = \sum_{j=0}^{k_i} c(v_j^i, v_{j+1}^i) + \sum_{j=1}^{k_i} \delta(v_j^i)$ is now (re)defined as the **cost** of route R_i.

1.3 Variants

There are many variants of the Vehicle Routing Problem that require a modification of the definitions given in the previous section. This section gives an overview of the the most common simplifications of, and extensions to, the VRP. Note that these variants do not necessarily exclude each other, combinations of two or more of these variants can be made to form more complex variants of the VRP. We will first consider some “small” variants of the Vehicle Routing Problem, before we dive into various subsections that each describes a variant.

Variants of the Vehicle Routing Problem with multiple depots are referenced to as **Multiple Depot** VRP’s. A VRP is considered to be a **Periodic** VRP (PVRP) if the planning is not done in one day, but spread over multiple days. A variant of the VRP that has a certain bound T on the maximum distance a vehicle travels each day is called a **distance constrained** version of the Vehicle Routing Problem. The total distance traveled often has influence on the total time (along with the service time). When a maximum allowed time per vehicle or for the entire routing is specified, we talk about the **time constrained** VRP.

We talk about the **Symmetric** Vehicle Routing Problem (SVRP) if the direction in which we travel from customer i to j does not matter, so $c_{ij} = c_{ji}$ for all customers i and j. If this is not the case and there exist customers i and j for which $c_{ij} \neq c_{ji}$, we talk about the **Asymmetric** Vehicle Routing Problem (AVRP).

1.3.1 Capacitated Vehicle Routing Problem (CVRP)

The **Capacitated** Vehicle Routing Problem (CVRP) is the most common and basic variant of the Vehicle Routing Problem. In this variant, a fixed fleet of m ($m \geq 1$) delivery vehicles must service the customer demands, with the additional restriction that these vehicles are **capacitated**: they can contain goods (the customer's demands) up to a certain maximum capacity.

In the **Homogeneous** CVRP (or **Uniform Fleet** CVRP) each vehicle in the fleet has the same capacity Q. The only difference in the formal definition is that a route is considered feasible if the total demand of all customers on a route R does not exceed the vehicle capacity Q: $(\sum_{j=1}^{k} d_j) \leq Q$ (where d_j is the demand of customer v_j). Of course, to ensure that vehicles are always big enough, the demand of a customer is never greater than the capacity of a vehicle: $d_j \leq Q$ $(1 \leq j \leq n)$. Also, the total demand of all customers can not be greater than the total capacity of all vehicles: $(\sum_{j=1}^{n} d_j) \leq m * Q$.

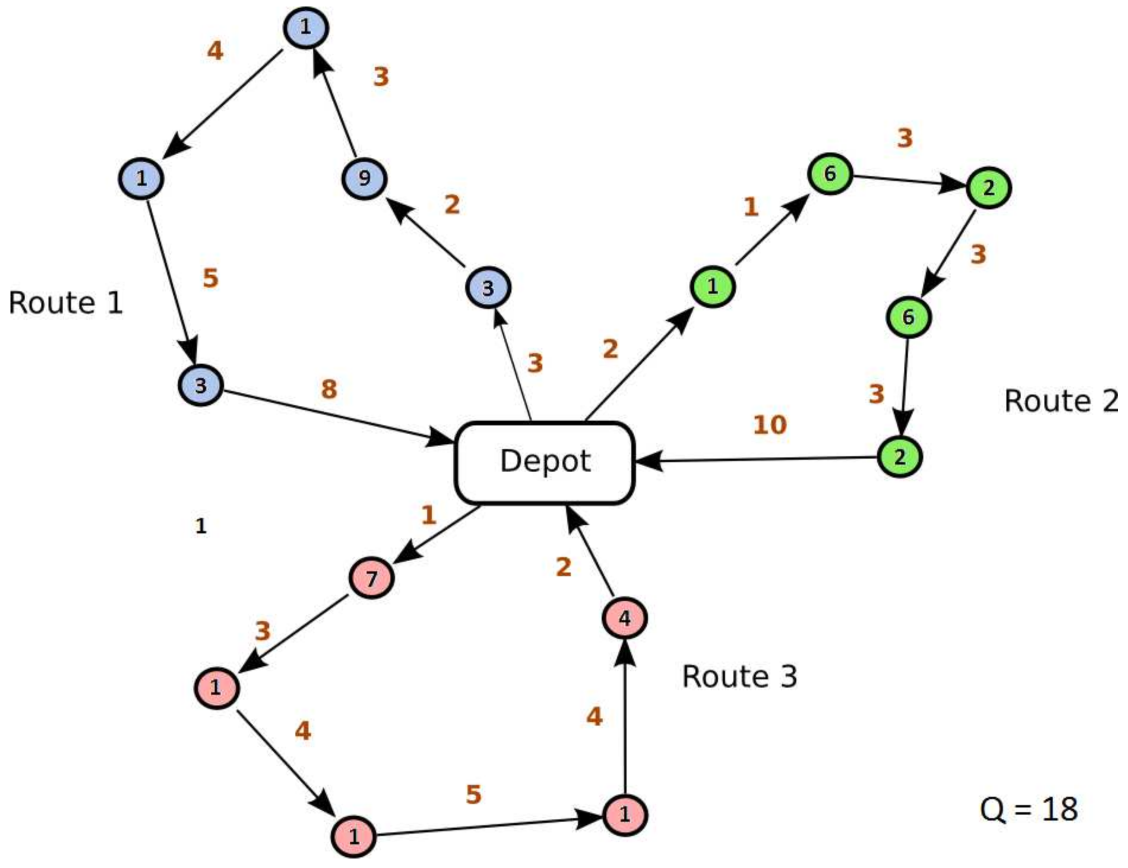


Figure 1.2: An example of a solution of the Capacitated Vehicle Routing Problem with with 1 depot, 15 customers, 3 vehicles, customer demands (given inside the nodes) and a vehicle capacity $Q = 18$. The three vehicles 1, 2 and 3 used capacity 17, 17 and 14, respectively.

In the **Heterogeneous** CVRP (or **Mixed Fleet** CVRP) the fleet is composed of different vehicle types, each with its own capacity Q_f $(1 \leq f \leq m)$. A route is now defined by a tuple (R, f), where R is the route vector and f the vehicle type. Restrictions similar to the ones we defined for the homogeneous CVRP apply for the maximum demand per route and the maximum total demand in relation to the capacity of the vehicles(s).

In some variants a type of vehicle also comes with its own fixed cost F_k, for example rental costs. Even the cost matrix C can be specific for each vehicle, so that we speak about a routing cost of c_{ij}^k for getting from customer v_i to customer v_j with vehicle k.

The Capacitated Vehicle Routing Problem is a problem which, beautifully formulated by [16], "lies at the intersection" of the earlier mentioned TSP and the **Bin Packing Problem (BPP)**. This problem is about objects of different volumes that must be packed into a finite number of bins of a certain capacity, minimizing the number of bins used. The BPP is actually an instance of the VRP with one depot, a homogeneous fleet and a cost matrix that has $c_{ij} = 0$ for all customers i and j. We refer the reader to [38] for more details on the Capacitated VRP.

1.3.2 Vehicle Routing Problem with Time Windows (VRPTW)

In the Vehicle Routing Problem **with Time Windows** (VRPTW), there is an additional restriction that states that customer v_i has to be served within a specific time window $[e_{v_i}, \ell_{v_i}]$. This interval is always within the **scheduling horizon**: the bounds of the time window of the depot: $[e_{v_0}, \ell_{v_0}]$. The definition of the feasibility constraints of the VRP is extended with the notion that each customer should be served within the bounds of its defined time window.

We have seen how the Vehicle Routing Problem is related to the TSP and the BPP. Adding time windows to the VRP introduces a third subproblem into the VRP, the **Job Scheduling Problem**. For this variant the goal is to schedule multiple jobs of different length onto one or more machines, attempting to minimize the length of the total schedule. An excellent overview of the difference between the VRP and the Job Scheduling Problem is given in [4].

1.3.3 Vehicle Routing Problem with Pickup and Delivery (VRPPD)

The Vehicle Routing Problem **with Pickup and Delivery** (VRPPD) is a variant in which the possibility that customers return some commodities is contemplated, so it is necessary to take into account that the goods that customers return to the delivering vehicle must actually fit into that vehicle. Again several variants have been proposed. For example, there could be a restriction that all picked up goods are returned to the depot, and there is no interchanging of goods between the customers. However, a rental company (for example where you could rent a printer) could remove this restriction. This is sometimes referred to as the Vehicle Routing Problem **with Mixed Service** (VRPMS).

A very similar variant is the Vehicle Routing Problem **with Backhauls** (VRPB), which is in essence the same as the VRPPD, but with the critical restriction that all goods must be delivered before any goods can be picked up, because the vehicle is filled "Last-In, First-Out" (LIFO). We refer the reader to [14] for more information about this version of the VRP.

1.3.4 Split Delivery Vehicle Routing Problem (SDVRP)

In the **Split Delivery** Vehicle Routing Problem (SDVRP), the restriction that each customer is served at most once is removed. For some instances of the VRP, this relaxation can introduce interesting savings in terms of total cost. This variant also allows the customer to demand more of a good than the capacity of one vehicle, a situation that is not unrealistic in real life. Finding a solution to an instance of this variant of the VRP is much more complex, and could theoretically make it a continuous optimization problem instead of a discrete optimization problem. The SDVRP is studied in great detail in [1].

1.3.5 Stochastic Vehicle Routing Problem (SVRP)

The **Stochastic** Vehicle Routing Problem (SVRP) covers all the variants of the VRP in which one or more properties of the VRP are random. For example, the customer can be present only with a certain probability (think of an ice-cream car looking for children to serve). It can also happen that customers have a certain random demand (for example depending on whether or not the children are hungry). A random factor could also be incorporated in the service time (i.e., is it raining while goods are unloaded, slowing down the process?). Even more dynamic is the variant where the distance matrix is not static and has random factors influencing it. The latter version is interesting, as it has a nowadays more and more important practical application: traffic congestion situations. A good view on this variant of the VRP is given in [39].

Figure 1.3: Left: Site-Depencendy: big trucks cannot enter small streets (SDVRP, Section 1.3.6). Right: Traffic congestion, an issue in the Stochastic VRP (SVRP, Section 1.3.5).

1.3.6 Site-Dependent Vehicle Routing Problem (SDVRP)

The **Site-Dependent** Vehicle Routing Problem (SDVRP) is a variant of the Heterogeneous Capacitated VRP where there exists a dependency between the type of vehicle and the customer, meaning that not every type of vehicle can serve every type of customer because of site-dependent restrictions. For example, customers located in very narrow streets cannot be served by a very big truck, and customers with very high demands require large vehicles. So associated with each customer is a set of feasible vehicles. This variant is discussed in, for example, [23].

1.3.7 Arc Routing Problem (ARP)

An item that strictly does not fit in the list of variants given in the previous subsections is the variant in which customers are located at arcs of the graph instead of at the vertices. This problem is called the **Arc Routing Problem (ARP)** and is addressed in great detail in [17]. Interesting to notice is that is has been shown by [32] that any Arc Routing Problem with n vertices can be converted into an equivalent Vehicle Routing Problem with $3n+1$ vertices (with some additional quite common requirements even into an equivalent VRP with $2n+1$ vertices, as shown by [28]). A very good survey of this problem is given in [42].

1.4 Complexity and Hardness

As mentioned before, the Vehicle Routing Problem definition lies somewhere at the intersection of the definition of the Traveling Salesman Problem (TSP) and the Bin Packing Problem (BPP). Both of these problems are NP-hard, which does not make it hard to believe that the VRP is also NP-hard. We have seen that we can easily reduce the TSP to the VRP: just take an instance of the VRP with one depot, one vehicle with an unlimited capacity (or set all demands to zero), a cost function proportional to only the distance, and an arbitrary number of customers (cities). We can even remove the restriction of one vehicle, which reduces the problem to the Multiple TSP which is also NP-hard. Similarly we can reduce the BPP to the VRP by considering the variant of the VRP with one depot and a cost matrix of all zero's.

So we know that the VRP lies roughly at the intersection of these two other problems, but how much harder does that make the VRP? Let us look at some numbers. The smallest unsolved VRP instance ("unsolved" means that the best set of routes has not yet been found) has 50 customers and 8 vehicles (`B-n50-k8` from [16]). Solving this instance, ignoring the capacity constraints and thus treating it as a Multiple Traveling Salesman Problem, takes less than a second. Finding a valid Bin Packing Problem solution also does not take more than a second. However, the optimal VRP solution has not yet been found. Experiments suggest that the VRP is not just two times harder than the other two problems, it is at least polynomially or maybe even exponentially harder than the two problems on which it is based.

We will not go into further detail on the complexity of the VRP, but this section shows us that we have an extremely difficult yet challenging task at hand. For more information about the complexity of the Vehicle Routing Problem, see [24].

Chapter 2

Solving Methods

2.1 Introduction

Many solving methods have been proposed for the CVRP. We distinguish between **Exact Methods**, methods based on **Heuristics**, and those based on **Meta-heuristics**. In this chapter, we will give examples of these approaches and explain several algorithms that have been developed for the Vehicle Routing Problem over the past five decades.

The rest of this thesis focuses on the Symmetric Homogeneous Capacitated Vehicle Routing Problem with one depot, in the future addressed as the Capacitated Vehicle Routing Problem (CVRP), or even shorter for reader convenience, "**the VRP**".

2.2 Exact Methods

There are no exact algorithms that work well enough to solve every instance of the VRP (hence studying the VRP is still very interesting!). So why does this section exist? There are some methods that work well up to a certain amount of customers and vehicles. Of course there are also particular situations in which finding a good solution is easy even for larger numbers of customers and vehicles. We will however focus on the general case in which the customers are more or less randomly distributed over some geometric space. We assume that extremely easy (or extremely hard) situations do not occur more frequently than one would expect from a random distribution.

The easiest exact method that one could think of is a simple **brute-force** approach. The most simple constructive approach would start with all empty routes and repeatedly extend the current route, at each node of the search tree either finishing the current route and proceeding with the next one, or selecting some customer to be visited next. In essence we are listing our n customers in some order (which can be done in exactly $n!$ ways), and we then place $m-1$ delimiters that determine when a route has ended after $m-1$ out of the $n-1$ (placing it after the last customer creates an empty vehicle) customers, which can be done in exactly $\binom{n-1}{m-1}$

ways, creating a total of $\frac{n!\binom{n-1}{m-1}}{m!}$ possible solutions (we divide by $m!$ because the order of the vehicles is irrelevant). One can imagine that with more than 10 customers and 3 vehicles, this method will soon be way too complex. Even though there are smarter ways of implementing a solver for the VRP with a brute-force strategy, because the problem is NP-hard, it is highly unlikely that we will ever find the optimal solution with a brute-force approach. We will have to try to be a little smarter.

Branch-and-bound is a search algorithm that at each node of the search tree evaluates the child nodes, assigns some bound to each node, and repeatedly selects the node with the best bound found so far for expansion. When we deal with a maximization problem, we use an upper bound, and analogously when handling a minimization problem such as the Vehicle Routing Problem, we talk about a lower bound. The process is often implemented using a priority queue, that initially contains the root node of the search tree. Then the first node in the priority queue is expanded, after which its children are evaluated, and added to the queue. We repeat this process until the next node in the queue has a worse bound than the best actual solution found so far, and thus we have found the best solution, as all the remaining nodes will ultimately result in a worse solution.

A branch-and-bound algorithm for the VRP clearly requires a lower bound, because we are trying to minimize the total cost. Over the past 50 years, many lower bounds have been suggested for the Vehicle Routing Problem. An excellent survey of lower bounds is given in [3]. In [22], a description of a branch-and-bound algorithm for the VRP is given. This algorithm converts the Vehicle Routing Problem into a so-called "K-tree", a structure for which a polynomial algorithm exists to find shortest paths. Amongst other smart things, this algorithm partitions the problem by fixing the edges between certain clustered customers. Some side constraints that take care of the vehicle capacity and the fact that each customer is visited at most once are also added. This algorithm has produced proven optimal solutions for a number of difficult problems, including a well-known problem with 100 customers. However, it still leaves certain 50-customer problems unsolved. No exact algorithms performing better than this one have been found in literature.

Many other exact methods have been proposed in literature. Because these exact methods are rather limited in performance compared to the heuristic methods, and this thesis focuses mostly on the application of Monte Carlo Techniques with which bigger instances can be solved, we will not give an extensive overview of all the other existing exact algorithms. We refer the reader to [26] and [8] for an excellent overview of exact algorithms for the VRP.

2.3 Heuristic Methods

In this section we will describe several heuristic methods, of which many exist. We will focus on the more AI-oriented methods, that can be roughly categorized into:

1. **Construction algorithms**: start with an empty route and extend it gradually, keeping the total cost as low as possible.

2. **Savings-based algorithms**: start with a lot of small routes and combine them as long as this improves the solution ("combine and conquer").

3. **Tour Splitting Algorithms**: start with one big tour and split it into capacity-feasible subtours.

In the next three subsections we will give examples of each of these algorithms.

2.3.1 Nearest Neighbor Insertion (NNI)

The **Nearest Neighbor Insertion (NNI)** approach is relatively simple and was introduced in 1983 in [5]. It perhaps is the most straightforward (and greedy) **construction algorithm** for the VRP one can think of, and works as follows. Initially all of the to be generated routes are empty. Starting with the first route/vehicle, until this current vehicle is full, we keep inserting the nearest unvisited customer as long adding this customer does not exceed the capacity of this vehicle. Then we select the next vehicle, and repeat the above, until either all the vehicles are full or until all customers have been served. When the algorithm has ended, we have either served all the customers, or there are still some customers left that have not been served. The latter can happen more frequently if the tightness of the problem is very high.

The **tightness** T of a VRP is defined as the relation between the sum of the demands of all customers and the total capacity of all the vehicles,

$$T = \frac{\sum_{i=1}^{n} d_i}{Q * m}$$

If this value is close to 1, there is a big chance that the NNI method does not give a satisfying result. A formal outline of the NNI method is given in Algorithm 1. This algorithm returns an array of routes that start and end at the depot. The function *addToRoute(i)* that can be called on a route adds customer i to the end of that route, whereas the function *nearestUnvisitedCustomer(i)* returns, trivially, the unvisited customer closest to customer i. The returned solution $routes[m]$ may not have visited all customers because of the earlier stated problem with a high tightness value.

Sometimes the problem of not having served all customers can be solved by not proceeding to the next vehicle when the nearest unvisited customer has a too large demand for the current vehicle, but to consider the entire list of unvisited customers. This may result in a feasible solution, but will often greatly reduce the solution quality. It could happen that the only capacity feasible unvisited customer is on the other side of the map, introducing a very long route length, whereas this customer could easily have been served later on by another vehicle that was in the neighborhood anyway. Another option to solve this problem is to perform a quick search when the algorithm has finished to try and insert the (often very few) unvisited customers into routes of vehicles that have some space left.

Let us look at an example of a VRP instance that we are going to solve with the NNI method. Suppose we have two vehicles ($m = 2$) of capacity 10 ($Q = 10$), 4 customers ($n = 4$), so we have v_1, v_2, v_3 and v_4 and of course the depot v_0. The four customers have demands $d_1 = 1$, $d_2 = 1$, $d_3 = 9$ and $d_4 = 9$. Suppose customers v_1 and v_2 both have a very small distance to each other and to the depot, and v_3 and v_4 a very large distance. The NNI method will first choose v_1 and then v_2 to be served by the first vehicle. Now we have route vectors $R_1 = (v_1, v_2)$

Algorithm 1 NEAREST NEIGHBOR INSERTION (NNI)

Require: n, m, $distance[n][n]$, $demand[n]$, Q

Ensure: $routes[m]$

```
 1: $capacity[m] \leftarrow array()$;
 2: $routes[m] \leftarrow array()$;
 3: $j \leftarrow 0$;
 4: while $j < m$ do
 5:     $current \leftarrow$ nearestUnvisitedCustomer(0);
 6:     while $capacity[j] + demand[current] \leq Q$ do
 7:         $routes[j]$.addToRoute($current$);
 8:         $capacity[j] \leftarrow capacity[j] + demand[current]$;
 9:         $current \leftarrow$ nearestUnvisitedCustomer($current$);
10:     end while
11:     $j \leftarrow j + 1$;
12: end while
13: return $routes[m]$;
```

and $R_2 = ()$, leaving a capacity of $10 - 1 - 1 = 2$ for the first vehicle and 10 for the second. However, the customers still to be served, v_3 and v_4, both have a demand of 9, but we no longer have two vehicles available with a remaining capacity of 9, and the definition of the CVRP (see Section 1.3.1) does not allow split delivery (see Section 1.3.4). If split delivery were allowed, the NNI algorithm would always solve the problem regardless of the amount of customers, their demands and vehicle size, even if the tightness is exactly 1. The main problem here is that the NNI method in all its greediness tends to forget about the Bin Packing Problem component of the Vehicle Routing Problem, but only relies on the distance between the customers. Another problem is, again as a result of this greedy approach, that the way that is traveled back to the depot is not taken into account. When a vehicle is at its capacity, the last visited customer But as long as the tightness is not too high, the NNI method can give a feasible and quite reasonable solution of a CVRP instance (see Chapter 4 for results).

2.3.2 Clarke & Wright's Savings Algorithm (CWS)

The **Clarke & Wright's Savings (CWS)** algorithm is older than the previous one and dates back to 1964 when it was introduced in [9] as the first **savings-based algorithm** (sometimes also refered to as **merging-algorithm**). This method initially assumes that each customer is served by its own vehicle. Next, two customers are to be served by the same vehicle as long as their capacity constraints are not violated. Determining the order in which customers are combined into a certain vehicle route is done by calculating the savings for a pair of customers:

The **savings** s_{ij} for a pair of customers v_i and v_j is defined as the savings in terms of distance that would be realized if these two customers would be served right after each other by the same vehicle instead of each by their own vehicle.

$$s_{ij} = c_{0i} + c_{0j} - c_{ij}$$

Note that, due to the triangle inequality, this quantity s_{ij} is larger than or equal to 0. An outline of the CWS algorithm is given in Algorithm 2. The algorithm takes the savings list, sorted in descending order, and processes the customer pairs on the savings list if they satisfy conditions a, b or c, assuming that operation does not violate capacity constraints.

The algorithm has a parallel and a sequential variant. The difference between the two is that the parallel version builds multiple routes at a time (this variant is outlined in Algorithm 2), whereas the sequential version builds one route at a time. In the parallel version it can happen that, when the savings list has been processed, unassigned customers are assigned to their own vehicle, exceeding the total amount of available vehicles m. In that case again, just like with the NNI method, a local search could be performed to serve these customers by one of the other vehicles.

Algorithm 2 CLARKE & WRIGHT'S SAVINGS (CWS)

1. Calculate the savings for every pair of customers.
2. List the previously calculated savings in descending order of magnitude, creating the "savings list."
3. Then for each savings pair s_{ij} on the savings list, include path (i, j) in a route if no capacity constraints will be violated through the inclusion of $i - j$ in a route, and if either one of the following statements holds:
   (a) Neither i nor j have already been assigned to a route, in which case a new route is initiated including both i and j.
   (b) Exactly one of the two points (i or j) has already been included in an existing route and that point is not interior to that route (a point is **interior** to a route if it is not adjacent to the depot in the order of traversal of points), in which case the link (i, j) is added to that same route.
   (c) Both i and j have already been included in two different existing routes and neither point is interior to its route, in which case the two routes are merged.
4. If the savings list has not been exhausted, return to step 3. Otherwise the algorithm terminates and the solution to the VRP consists of the routes created so far. If any unassigned customers remain, they must be served by their own vehicle.

Let us look at an example. Consider the symmetric distance matrix in Table 2.1 for 5 customers ($n = 5$) and demand vector given in Table 2.2. Assume that we have 2 vehicles available ($m = 2$) and the capacity Q is equal to 100. We will outline how both the sequential and the parallel version processes this example.

We first compute the savings of all the customer pairs v_i and v_j by applying the previously mentioned formula to the distance matrix. The result is shown in Table 2.3.

c_{ij}	v_0	v_1	v_2	v_3	v_4	v_5
v_0	0	28	31	20	25	34
v_1		0	21	29	26	20
v_2			0	38	20	32
v_3				0	30	27
v_4					0	25
v_5						0

Table 2.1: Symmetric distance matrix for a VRP with 5 customers ($n = 5$).

Customer	Demand
1	37
2	35
3	30
4	25
5	32

Table 2.2: Demand vector for the 5 customers from the VRP in Table 2.1.

s_{ij}	v_1	v_2	v_3	v_4	v_5
v_1	0	38	19	27	42
v_2		0	13	36	33
v_3			0	15	27
v_4				0	34
v_5					0

Table 2.3: Calculated savings based on distance values of Table 2.1.

For convenience we sort the pairs of customers of Table 2.3 by savings, in descending order, creating the savings list:

$1 - 5$
$1 - 2$
$2 - 4$
$4 - 5$
$2 - 5$
$1 - 4$
$3 - 5$
$1 - 3$
$3 - 4$
$2 - 3$

Let us start with the sequential variant. Customers 1 and 5 are considered first. They can be assigned to the same route since their joined demand for 69 units does not exceed the vehicle capacity of 100. Now we establish the connection $1 - 5$, and thereby points 1 and 5 will be neighbors on a route in the final solution. Next we consider customers 1 and 2. If customers 1 and 2 should be neighbors on a route, this would require the customer sequence $2 - 1 - 5$

(or $5-1-2$) on a route, because we have established already that 1 and 5 must be visited in immediate succession on the same route. The total demand (104) on this route would exceed the vehicle capacity (100). Therefore, customers 1 and 2 are not connected. If points 2 and 4, which is the next pair in the list, were connected at this stage, we would be building more than one route ($1-5$ and $2-4$). Since the sequential version of the algorithm is limited to making only one route at a time, we disregard $2-4$. The combination of the next pair of points, 4 and 5, results in the route $1-5-4$ with a total demand of 94. This combination is feasible, and we establish the connection between 4 and 5 as a part of the solution. Running through the list we find that due to the capacity restriction no more points can be added to the route. Thereby we have formed the route $0-1-5-4-0$. In the next pass of the savings list we only find the point pair 2 and 3. These two points can be visited on the same route, and we make the route $0-2-3-0$. The sequential algorithm has constructed a solution with two routes. The total cost for the route $0-1-5-4-0$ is 98, and for the route $40-2-3-0$ the total cost is 89, which makes a total cost of 187.

Now consider the parallel version of the algorithm which may build more than one route at a time. In this version 1 and 5 are also combined first because they have the highest savings. Points 2 and 4 are now also combined in the second route. We now have routes $0-1-5-0$ and $0-2-4-0$. Only Customer 3 is now left and gives the highest savings with customer 5, so it is added to the first route. In this way the algorithm constructs the routes $0-1-5-3-0$ and $0-2-4-0$ with a total cost of 171. In this case the parallel version performed better (171 compared to 187).

The advantage of the parallel method is that the savings list has to be processed only once, and the result is on average better than that of the sequential variant. Therefore, in the future, when we mention the CWS method, we refer to the parallel variant.

The CWS algorithm has very often been adjusted, improved and tuned, for example in [21]. As we will see later in this thesis, the algorithm is a very good basis for many other algorithms, because of its nice complexity of $O(n^2 \cdot log\ n)$ with $n \geq 1$ customers (when implemented as a heap with n^2 elements and extraction from the heap takes $log\ n$ time).

2.3.3 Tour Splitting Algorithms (TSA)

Tour Splitting Algorithms (TSA) for the CVRP start by building one giant tour visiting all customers. This tour is then split into capacity-feasible vehicle routes. This method is seldom used alone because of a reputation of limited performance. However, [33] has successfully implemented an algorithm of this kind that can efficiently solve instances of the Vehicle Routing Problem. The problem lies not only in the splitting of the giant route, but just as much in finding a good giant route. As one might expect, the TSP-optimal giant route is often not the best route to go with, because there is also the BPP element to take into account.

In [33] several methods to generate a route are mentioned. We will outline one of these methods called **RTF (Random Task Flower)**, named after the visual pattern of a flower (see Figure 2.1 than arises when the giant tour has been generated. Important are two lists into which the candidates are partitioned after each step, L_1 and L_2. L_1 gathers the unvisited customers which drive the vehicle away from the depot, and the other unvisited customers are stored in L_2. The total cost is kept in a variable r, initialized at 0. If the current route is empty, RTF selects one customer at random. If the route is not empty, RTF randomly draws the next customer from L_1 if $(r \ mod \ Q) < Q/2$ and from L_2 otherwise. The emerging behavior here is that the giant tour tends to go away from the depot if it is less than half full, otherwise it gets closer to the depot. A splitting algorithm is more likely to cut the sequence when the vehicle is close to the depot, so this helps the splitting.

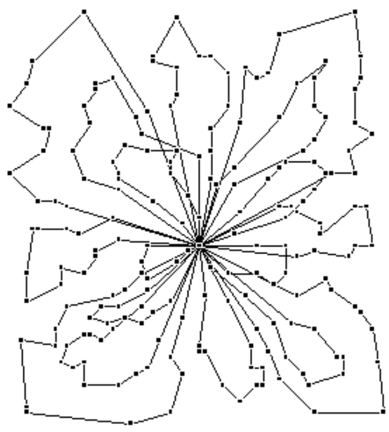


Figure 2.1: Flower pattern by "Random Task Flower"-algorithm.

After the giant tour has been generated, for example by the method described above, it is split using a splitting algorithm. In the end a local search has to be performed to improve the final solution. The results of this Tour Splitting approach are comparable to those of meta-heuristics, but this method is claimed to be faster and simpler. We refer the reader to [33] for an an excellent overview.

2.4 Meta-heuristics

In this section we describe general meta-heuristics that can and have been applied successfully to the Vehicle Routing Problem.

2.4.1 Ant Colony Optimization

Ant Colony Optimization (ACO) is a class of optimization techniques belonging to the field of Natural Computing. It was introduced roughly 20 years ago by [15] and uses a solution method that resembles the way in which real ants behave when searching for food in an environment. During their search they mark the trails they are using by laying down **pheromones**. The amount of pheromone on a certain path lets other ants know whether or not it is a promising path. The general algorithm works as follows.

Each iteration of the ant colony algorithm consists of all ants building (part of) a solution of the problem step by step. Which steps are made by the ants depend on the amount of pheromone on that trail, as well as on the attractiveness of that particular move. This attractiveness is a domain-specific property. After the solution has been built, the amount of pheromone on each trail is updated according to some update rule, and the process is repeated. When tuned properly, in the end, all ants should be walking along the optimal path.

In [29] several components of an ACO algorithm for the CVRP are outlined. The first component is the **routes building** which determines how each ant builds the solution. In the parallel version, each ant designs the route for all vehicles at the same time. At each iteration of the algorithm only one client is chosen. The best tour is then extended. When each ant builds an entire solution composed of multiple routes we talk about the sequential version. The steps of the ant towards a customer are of course based on the amount of pheromone on that trail, and on the attractiveness, which is a function that depends on the length of the path between the current and candidate customer. A tabu list of the already visited customers is also kept. The second component of the ACO algorithm is the **transition rule**: the rule that defines which customer is chosen next by a certain ant. A choice can be made between "exploration" (select a random customer, or perhaps a random customer proportional to the distance to that customer) and "exploitation" (select the customer on the path with the highest level of pheromone). The third important component is the **pheromone actualization**, which can be done while a solution is built, or after the solution has been built. A frequently used actualization technique for the parallel version is the "Elite Ant"-technique, where only the pheromone on the parts of the route that belong to the best solution are updated. Several more domain-specific components are also outlined in [29], to which we also refer the reader for more information on the ACO technique, which performs better than for example the CWS Algorithm.

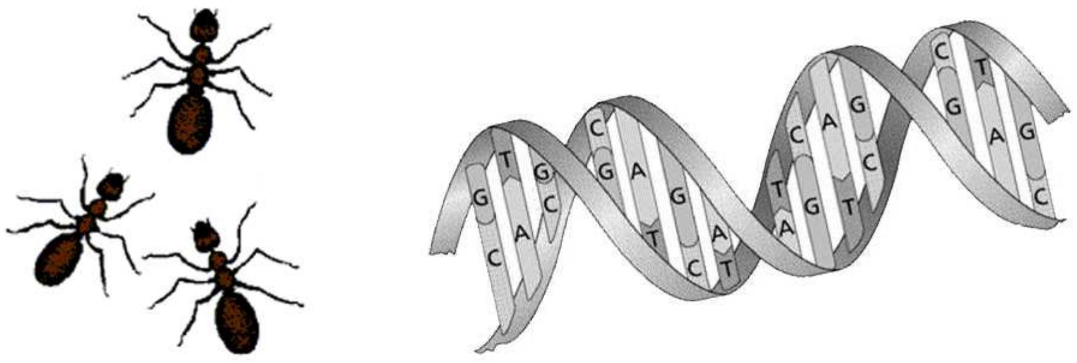


Figure 2.2: Ant Colony Optimization (ACO) and Genetic Algorithms (GA's).

2.4.2 Genetic Algorithms

A **Genetic Algorithm (GA)**, for example as described in [18], is a form of reinforcement learning that, just like ACO, belongs to the field of Natural Computing. The idea behind a genetic algorithm is that there is a population of candidate solutions (the **individuals**) that, by repeatedly applying **mutations** and **crossovers**, will evolve to a population with better (and ultimately optimal) solutions. The quality of an individual is evaluated by a **fitness** function. A genetic algorithm works as follows.

First a population of solutions is, for example randomly, initialized. Then, while some stopping criterion (a fixed number of iterations, or reaching a certain solution quality) is not satisfied, two parents are selected and a crossover is done to create new children. Next, each of these new children has a certain chance of receiving a mutation. The children are then evaluated as well. In the last step the new population is determined, where survival of individuals in the population depends on the fitness value(s). Many variations of GA's exist. If for example the new population consists of only the children, we talk about a **Generational GA**, and when the new population consists of a selection of the best individuals (so both parents and children), we talk about a **Steady State GA**.

In order to use a GA to solve the Vehicle Routing Problem we have to define how we can express a solution of the VRP as an individual. We could simply use a string that consists of a concatenation of the customers of all the routes. This representation is however not very flexible when we want to apply the different genetic operators (see next paragraph). Therefore [2] suggests representing the solution/individual as a string of length n ($n \geq 1$, the number of customers), where each character in the string has a value x with $1 \leq x \leq m$ ($m \geq 1$, the number of vehicles). This however does not explicitly define the order in which the customers are served. For that, a TSP solution will have to be found for each vehicle. The fitness of a solution can be determined by the total cost of the routes, incremented by a huge penalty when the routes are not capacity-feasible. Several steps are taken to ensure that the GA can do its work as efficiently as possible. One of them is sorting the customers in the string so that consecutive customers are likely to be served by the same vehicle. For problems where the customers are randomly distributed around the depot, they are sorted by their distance to the depot in increasing order. When customers are located in clusters, they are sorted according to a simple nearest neighbor TSP structure, starting from the depot and visiting all customers. The representation method described above has the advantage of never visiting customers more than once after a suitable crossover has been applied.

The next question is how to generate the initial population. Creating a population with only near optimal solutions, obtained by for example the CWS method (see Section 2.2) may not include sufficient diversity, whereas a completely random population will perhaps never converge towards the good solutions. As one may guess, a combination of the two will often do fine.

According to [36], there are three **genetic operators** can be used to tune a GA:

- **Selection**: How are the candidates for reproduction chosen? **Roulette-Wheel Selection** is often used. This method selects an individual with with fitness f with probability $f/(n * \bar{f})$, where $\bar{f}$ is the average fitness and $n > 0$ the size of the population. It is also possible to just select two parents at random.
- **Crossover**: How are parents combined? Many crossover strategies exist. For the VRP, [2] suggests using **2-point-crossover**, a method in which two crossover points (borders between two characters in the individual's string) are selected, dividing the string into three parts. The new string consists of the first and third part of one parent, and the second part of the other parent.
- **Mutation**: What kind of mutations are applied to the children? The most common mutation method applicable to the VRP is simply switching two randomly selected customers between vehicles with a certain probability.

In [2] it is shown that genetic algorithms, especially along with some domain-specific methods to "cure" unfit solutions, perform comparable to other meta-heuristics for the Vehicle Routing Problem. The results obtained are very close to optimal solutions and outperform methods such as CWS, performing comparable to for example Ant Colony Algorithms and Tabu Search.

2.4.3 Tabu Search

Tabu Search is a local search method that explores the solution space by moving from a certain solution x_t to a next solution x_{t+1} by taking the best solution out of a set of solutions in the neighborhood $N(x_t)$ of x_t, assisted by some memory. The initialization of the first solution, x_0, can be done at random, or can for example be the solution obtained by applying the CWS method from Section 2.3.2. The algorithm stops when it is either out of computation time, or when no better solutions are found for a certain number of iterations. Because x_{t+1} is not necessarily better than x_t, a **tabu mechanism** prevents cycling over the same solutions over and over again. Exactly implementing this often referred to as "short-term-memory"-mechanism is a too complex task, so instead certain attributes of past solutions are registered, for example using a smart hash function. Any new solution that matches one in the memory is not considered for a certain amount of iterations. Tabu Search is often strengthened by the so-called "long-term-memory", which consists of two features, diversification and intensification. **Diversification** is the process of ensuring that the search does not get stuck in local optima and **intensification** makes sure that the search occasionally focuses on one certain very promising part of the search space.

Tabu Search is very similar to Simulated Annealing, but differs in a way that Simulated Annealing has a cooling scheme to analyze the search space as thoroughly as possible and to ultimately achieve convergence, whereas Tabu Search uses its "long-" and "short-term-memory" to achieve these two goals.

In order to apply Tabu Search to the VRP, we must define a neighborhood, for which many suggestions have been done in literature. We will explain the one defined by [31] named **λ-interchanges**. This method generates a neighborhood by repeatedly exchanging up to λ customers ($\lambda \geq 1$) between two routes in x_t. These exchanges can be described by couples (λ_1, λ_2) (with $\lambda_1, \lambda_1 \leq \lambda$) that represent an operation where λ_1 customers are moved from route 1 to route 2, and λ_2 customers are moved from route 2 to route 1. For example, in 2-exchanges the following types of moves are possible: (2, 2), (2, 1), (2, 0), (1, 1), (1, 0) and their symmetric counterparts. Of course only capacity-feasible moves are considered. The swapped customer is inserted into its new route for example after or before its nearest neighbor already in that route, but better yet more complex methods have also been proposed. To implement the before mentioned "short-term-memory", it is common to prohibit reverse moves for a certain amount of iterations.

A big problem with these algorithms is complexity. Therefore when applying λ-exchanges, the value of λ is often restricted to 1 or 2 in order to limit the number of possibilities. One could also experiment with slightly larger values of λ and limit the size of the neighborhood to a certain maximum size. Many other methods to define the neighborhood have been suggested. For an overview of these methods, other neighborhood definitions and more "short-" and "long-term-memory" algorithms, see [10], an article in which Tabu Search is presented as the best heuristic method for the Vehicle Routing Problem.

Chapter 3

Monte Carlo Techniques

3.1 Introduction

Monte Carlo Techniques (or **Monte Carlo Methods**, or in short **Monte Carlo (MC)**) are a class of techniques or algorithms that rely on the use of random sampling to finally acquire a solution to a given problem. The name "Monte Carlo" was given to this class of techniques by John von Neumann and Stanislaw Ulam somewhere around 1946, when they suggested to some physicists that their problem could perhaps be solved by modeling the experiment on a computer using chance. Being secret, their work required a code name. Von Neumann chose the name "Monte Carlo". The name is a reference to the Monte Carlo Casino in Monaco where Ulam's uncle would often go to gamble. Interesting to see is that nowadays Monte Carlo algorithms are widely used in algorithms for playing casino games such as poker. Did von Neumann know back then in the 1940s that his algorithmic approach would find an application in the field from where he simply picked the name?

Figure 3.1: The Monte Carlo casino and the Formula 1 circuit Monte Carlo in Monaco.

The random sampling used in Monte Carlo Techniques can be used and implemented in many ways. We will discuss a few uses of it in this chapter. Note that we will only consider the cases in which discrete (so non-continuous) choices have to be made, and full information is available. We made this choice not because Monte Carlo does not perform well in continuous and/or stochastic environments (on the contrary!), but because the problem we are finally going to apply these techniques to (the Vehicle Routing Problem), has these properties.

3.2 Random sampling

Exhaustive approaches more or less enumerate all possible solutions, frequently by traversing the total search space in some way by means of an algorithm. A Depth First Search (DFS) is a classic example of such an algorithm. However, when the search space is too large, a DFS will often get stuck in the bottom leftmost part of the search tree, introducing little to no diversity in the obtained solutions except for perhaps a bit of diversity near the leaf nodes. Very often, crucial decisions in the search tree that determine the quality of a solution are not made only close to the leafs, but at various much higher levels of the search tree, that will never be reached again by the DFS algorithm.

Algorithm 3 Random Sampling

```
Require: current
Ensure: solutionQuality
1: while !isLeaf(current) do
2:     n ← numberOfChildren(current);
3:     i ← rand() % n;
4:     current ← current.child(i);
5: end while
6: return qualityOf(current);
```

Plain **random sampling** is a method that starts at the root node of a search tree and repeatedly picks a random child as a successor until it reaches a leaf node. This simple algorithm is outlined in Algorithm 3, which returns only the quality of the solution (but of course the solution itself could also be saved and returned). The algorithm should be called with *current* set to *root*, the initial state. The process of random sampling can be repeated a number of times to obtain a set of solutions, or to only keep the best solution after each iteration. The advantage of this method is that there is maximum diversity in the set of obtained solutions in a reasonable amount of time, eliminating the previously stated problem of only visiting a small part of the three. Of course this method still has disadvantages; depending on the domain of the problem, good solutions are often not randomly distributed over the search tree, but even though scattered throughout the search tree, often clustered together at some points.

The simple method of random sampling can actually be more effective than heuristic methods in cases when it is hard to derive a good admissible heuristic for a certain state in the search tree. In that case, an iteration of a few hundred thousand random samplings can give better solutions compared to methods such as best-first-search and other greedy methods that rely on heuristics. The same advantage goes for exact methods such as brand-and-bound algorithms in the case when the upper bound is a too big over-estimator. In that case the search will resemble Breadth First Search (BFS) and a good solution will most likely never be found, and random sampling is again more effective. So we can conclude that even though the method of random sampling may seem crude and ineffective, in some cases, it can be useful.

3.3 Monte Carlo Simulation (MCS)

Monte Carlo Simulation (MCS), for example as described in [27], is often used as a heuristic evaluation method to determine which of the children in the fringe is to be selected next as a successor node in some derivation. This approach is especially useful when it is hard to judge the quality of a partial solution before the entire solution has been generated. The process is explained in Figure 3.2, and works as follows. We start at the root node of the search space which is marked as the current node. From each of the $n \geq 1$ possible successors (children) of this current node we perform $r \geq 1$ (common values are around 1000, but at least a value comparable to the amount of children of that node is needed for a good average) random simulations until a leaf node of the search space is reached. So at each step of this algorithm a total of $n * r$ so-called **probes** are sent down towards the leaf nodes. The "best" child based on these random simulations is then selected next for expansion. The "best" node can either be the node with the highest score out of the r random probes, or it can be the node with the highest average out of the r probes. Which evaluation method is best depends on the type of problem and the behavior of the search space. When the current node is a leaf, an actual solution has been found and the method is terminated. An outline of this algorithm, specifically the variant where the node with the highest average score is repeatedly selected, can be found in Algorithm 4. This algorithm again only returns the quality of the obtained solution, but of course the path could also be saved and returned. Initially the algorithm is again called with *current* set to *root*.

Algorithm 4 Monte Carlo Simulation

```
Require: $current$, $r$
Ensure: $solutionQuality$
 1: while !isLeaf($current$) do
 2:     $bestSolution \leftarrow +\infty$;
 3:     $bestChild \leftarrow -1$;
 4:     $n \leftarrow$numberOfChildren($current$);
 5:     for $i = 0$ to $n - 1$ do
 6:         $sum \leftarrow 0$;
 7:         for $k = 1$ to $r$ do
 8:             $sum \leftarrow sum +$ Random Sampling($current$.child($i$));
 9:             if $sum/r < bestSolution$ then
10:                 $bestSolution \leftarrow sum/r$;
11:                 $bestChild \leftarrow i$;
12:             end if
13:         end for
14:     end for
15:     $current = current$.child($bestChild$);
16: end while
17: return qualityOf($current$);
```

The MCS method can be improved by performing a full search as soon as the size of the remaining search space below the current node allows this. Another simple improvement can be realized by keeping track of the best found solution so far during all the random probes, as this solution may not be identical to the actual solution that is finally found at the leaf node. Even better would be to at some point restart the search from the point where a **critical misdecision** has been made. This is a decision where in current node w at level i ($i \geq 1$, level

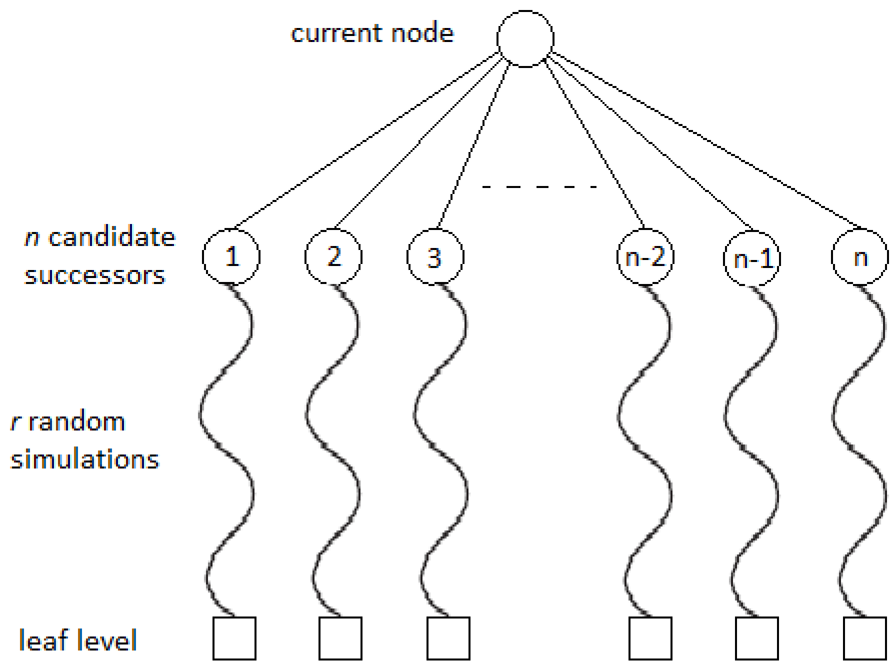


Figure 3.2: Basic Monte Carlo Simulation

0 is the root node) of the search tree node z at level $i+1$ is selected next, whereas instead in the previous current node x at level $i-1$ a path $w-x-y$ (with $y \neq z$) was found as the best solution. This decision to correct such a misdecision can occur after a number of steps (for example 5 or so) has been done that no longer improve the current solution, or it can be made at the end of the algorithm for each recorded misdecision.

The MCS method often performs much better than plain random sampling, because the search is “guided” towards a better part of the search tree where again better solutions can be found. Consider the part of a search space in Figure 3.3, regardless of the domain of this problem. Five optimal (green) solutions are more or less randomly spread over the search tree. A Monte Carlo Simulation algorithm will perform several random simulations and at some point find the orange areas which represent a part of the tree filled with better solutions. It will then focus on such an area, and try to find even “warmer” areas, hoping to ultimately find the green optimal solution. The performance of this procedure is however highly dependent on the size (or “width”) of the orange space. If an optimal solution is surrounded by a lot of good (orange) solutions, the orange and therefore in the end the green solution is likely to be found. However, if the green solution has only a handful of orange paths around it, these good areas are much less likely to be found. But, in essence, just like with a brute-force-algorithm, if there is enough computation time (in this case enough time to perform enough random simulations), the best solution will ultimately be found.

Other than just doing more random simulations, performance of Monte Carlo algorithms can also be improved by improving the simulation strategy. Instead of sending random probes down the search tree, domain-specific knowledge could be used to slightly guide the search. For example, in case of the Vehicle Routing Problem, it might not be smart to at some point create a path between two customers with a very large distance in between them, while both customers have

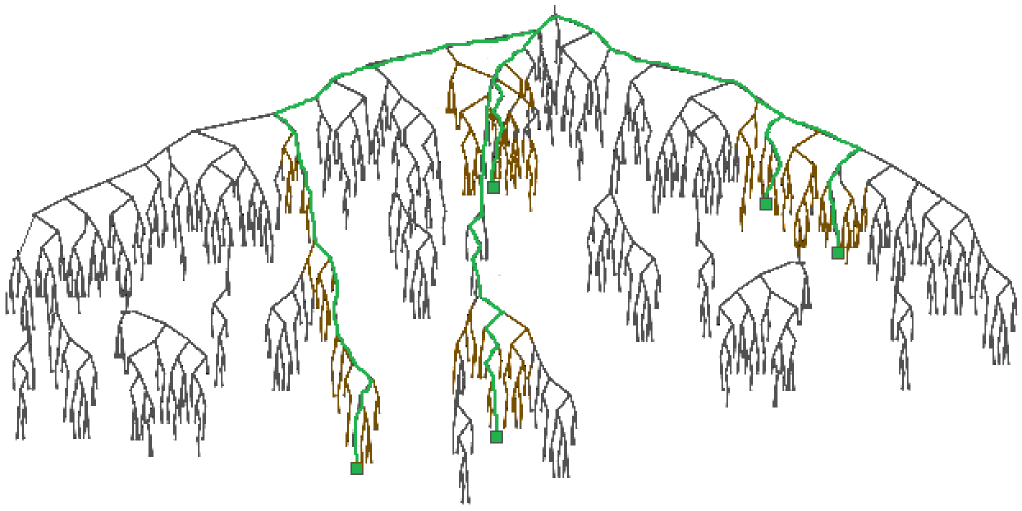


Figure 3.3: Part of a search space with good (orange) and optimal (green) solutions.

many other closer customers that they can be connected to. Therefore these kinds of decisions could be frequently avoided, as they are not likely to make the search end up in a good part of the search tree.

The problem with having a big random factor playing a role in algorithms is that even though we take an average out of a fixed (perhaps too small) number of random simulations, wrong choices can still be made. Therefore, one run of a Monte Carlo algorithm is not as efficient as ten runs. Often described in literature such as [37] as **Meta Search**, the so-called **restarts** can help improve the quality of the final solution. This can be done for example after a certain amount of time has passed, or when the search appears to be stuck in a local minimum.

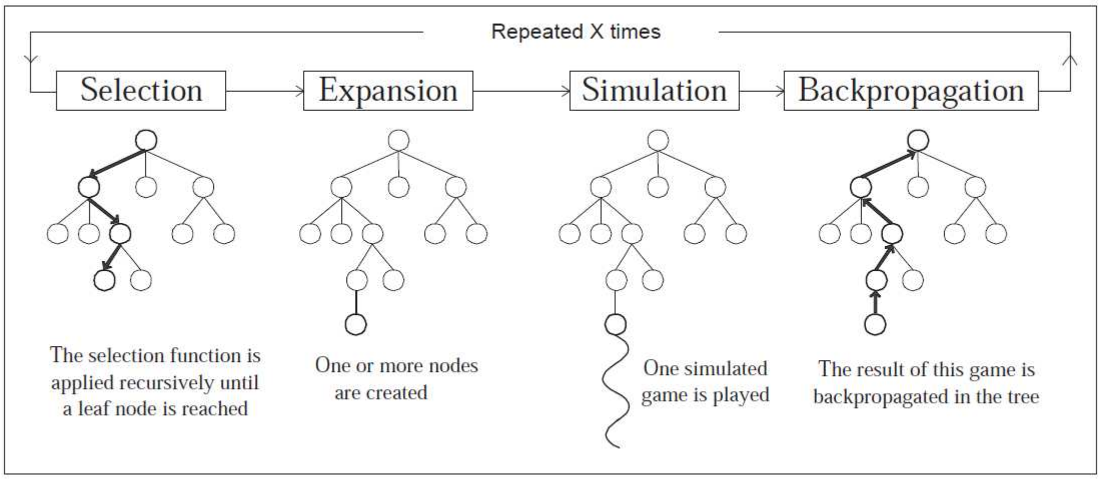
3.4 Monte Carlo Tree Search (MCTS)

Monte Carlo Tree Search (MCTS) is a more advanced Monte Carlo Technique and is described for example in [37]. The process is explained in Figure 3.4. A tree is built in memory, and the process that builds this tree has four important phases that are repeated until time runs out:

1. **Selection**: starting from the root, the algorithm traverses the already stored tree until it reaches a stored leaf. This is not necessarily a leaf of the full search tree (and therefore not yet a full solution).
2. **Expansion**: one or more nodes are added to the tree as child(ren) of the node reached in the previous step.
3. **Simulation**: one or more simulations are performed from the newly added node(s), down to a leaf of the search space. A heuristic can again be built into this, in case of the VRP we could for example decide to visit nearby customers with a higher probability. Here it

is important to not restrict the simulations and still allow them to do seemingly "dumb" transitions in order to keep the balance between exploration and exploitation intact.

4. **Backpropagation**: during this phase the result of the simulation is propagated back towards the root.



Repeated X times
Selection
Expansion
Simulation
Backpropagation
The selection function is applied recursively until a leaf node is reached
One or more nodes are created
One simulated game is played
The result of this game is backpropagated in the tree

Figure 3.4: Monte Carlo Tree Search

During the selection phase a choice of which node is going to be expanded next is made. On the one hand, the task is often to select the move that leads to the best results (the earlier mentioned **exploitation**) but on the other hand, the least promising moves still have to be explored, to ensure **exploration**). A commonly used method is Upper Bound Confidence Trees [25], a method that selects the node i with the highest value of:

$$v_i + C \times \sqrt{\frac{ln\ N}{n_i}}$$

Here v_i is the value of node i, n_i the visit count of i, and N is the visit count of the parent node of i. C is a coefficient, which has to be tuned experimentally. The value v_i of a node is computed by taking the average of the results of all simulated games made through this node.

Monte Carlo Tree Search is often used in the field of game theory, for example by [37] who used it in an attempt to solve SameGame puzzles. Several improvements of the existing method have been introduced, for example Nested Monte Carlo Tree Search which was suggested in [7].

3.5 Parallelization

Parallelization is the process of splitting up a task over multiple CPU's (or multiple cores) in order to increase computational power, and is often suggested as a speed-up method for algorithms. Since the introduction of multi core CPU's on the consumer market around 2005, a desktop PC nowadays has two, four six or maybe even eight cores, and this amount will only grow in the nearby future. Another recent development is that these days GPU's often come with a larger amount of cores (one hundred or more). We can easily conclude from these relatively recent developments that considering parallelization issues in solution methods becomes more and more important.

Monte Carlo Techniques can benefit greatly from parallelization. The random sampling of r samples at some point during for example Monte Carlo Simulation can be divided over $c \geq 1$ CPU's by simply giving each CPU r/c samples to compute. Depending on the type of problem we can also choose to assign a certain number of children to each CPU, so if we again have n candidate successors each CPU computes the random samples of n/c candidates. In case of MCTS, we can simply start the algorithm for each available CPU.

Chapter 4

Applying Monte Carlo Techniques to the Capacitated Vehicle Routing Problem

4.1 Introduction

This chapter is about the application of Monte Carlo Techniques to the Capacitated Vehicle Routing Problem. We will first consider some previous work, after which we will describe some new “Monte Carlo-inspired” derivatives of the search methods described in Chapter 2. We will compare the performance of these methods with existing work based on tests that we perform for each of the methods.

The tests are performed on the commonly used test sets from Christofides and Eilon and from Augerat et al., both found at [34]. The files of this test set are in the convenient TSPLIB [35] format. The obtained solutions, available in the Appendix, are also in the TSPLIB output format. For our experiments we used a 3.2GHz Core i7 quad-core machine with 6GB of memory and no more than 5 minutes of computation time.

4.2 Previous Work

Compared to the huge amount of work done on the Vehicle Routing Problem, relatively little work seems to have been done on applying Monte Carlo Techniques to the Capacitated VRP. We will give a brief description of the work found in literature.

Using a form Monte Carlo Techniques to solve the Vehicle Routing Problem was suggested for the first time in 1979 by [6]. Improvements compared to the CWS method were already observed at that time, though standard test sets were not yet defined, making comparison with current techniques quite hard. Random sampling for the distance constrained VRP was suggested in 2000 by [13]. A simulation method extremely similar to the NNI method was used, where customers are selected with a probability proportional to their distance to the last visited node. We will apply a similar strategy to the NNI method in the next section.

In 2007, the ALGACEA-1 method was introduced in [21] as a Monte Carlo algorithm for the VRP, based on Clarke & Wrights Savings approach and assisted by an entropy function. The method indeed performs better than the simple CWS method. SR-1 [19] is another a recent algorithm very similar to ALGACEA-1, also based on the CWS method, but unfortunately this algorithm was only tested on randomly distributed search spaces and not the well-known test sets from [34] or [40].

Monte Carlo Techniques have also been applied to CVRP-related problems. In [30] an algorithm based on Monte-Carlo-inspired randomization has been suggested for the uncapacitated time-constrained Vehicle Routing Problem, which is in essence a multiple Traveling Salesman Problem (MTSP) with a maximum distance per route and no additional capacity constraints. A Monte Carlo algorithm for the Rural Postman Problem (find the shortest closed route that visits every edge of a connected undirected graph) has been proposed in [12].

No other literature than the above was found on the application of Monte Carlo Techniques to the Capacitated Vehicle Routing Problem.

4.3 MCT applied to the NNI method

The Nearest Neighbor Insertion (NNI) method, a constructive method to solve the VRP, is described in Section 2.3.1. As explained, this method does not always give a feasible let alone optimal solution, especially when the tightness is very high (near a value of 1). We have applied several Monte Carlo Techniques, as described in Chapter 3, to the NNI method, and will report about our methods and results in this section.

Table 4.1 gives an overview of several test instances. The first column represents the name of the instance, where the two numbers stand for the number of customers (including the depot) and the number of vehicles, respectively. The second column denotes the tightness of that instance, followed by the best solution found so far in literature. For some instances, the exact best solution is not known, but only some very-near-optimal upper bound, denoted by $\leq$ in front of the value. A value is denoted between brackets if a capacity-infeasible solution was found for that instance.

The column ("100k-Random") gives the best score obtained when performing $100,000$ random simulations. In each step of this random sampling process a random choice of one customer out of either all the unvisited capacity-feasible customers is made, after which either or not the next vehicle is chosen. Selecting the next vehicle is not smart when the current vehicle is not very full yet, so the chance of selecting the next vehicle is proportional to how full the current vehicle is, so if the current vehicle is half-full there is a 50% chance of selecting the next vehicle. The performance of the "100k-Random"-method is, as one could expect, very bad. For smaller instances the difference with the optimal solution is not that big, but when the number of customers increases the method performs many times worse compared to the optimal solution, most likely because a much smaller part of the search tree is evaluated. Notice how the "100k-Random"-method often finds a solution that is far from optimal, but at least feasible. This method could most likely be improved by a smarter simulation strategy than the one described above, but we only added it to demonstrate how the search space is considerably larger (and the problem thus considerably more difficult) when the amount of customers and vehicles increases.

Instance	Tightness	Best	100k-Random	NNI	Random-MCS	NNI-MCS
`E-n13-k4`	0.76	247	271	339	259	258
`E-n22-k4`	0.94	375	515	472	500	411
`E-n23-k3`	0.75	569	921	(489)	921	785
`E-n30-k3`	0.94	534	941	598	877	598
`E-n31-k7`	0.92	379	1004	(422)	945	632
`E-n33-k4`	0.92	835	1388	(834)	1147	1095
`E-n51-k5`	0.97	521	1289	(639)	1217	1155
`E-n76-k7`	0.89	682	2035	869	1976	869
`E-n76-k8`	0.95	≤ 735	2084	(874)	1986	1501
`E-n76-k10`	0.97	830	2351	(842)	2330	1750
`E-n76-k14`	0.97	1021	(1861)	(984)	(2390)	2310
`E-n101-k8`	0.91	≤ 815	2780	1122	2542	1041
`E-n101-k14`	0.93	≤ 1071	2890	1287	2675	1180
`G-n262-k25`	0.97	≤ 6119	26745	(6176)	24441	21923

Table 4.1: Solution lengths for various NNI methods.

The NNI method is reported in the fifth column and often does not keep into account the Bin Packing Problem (BPP) element of the VRP and therefore often does not find a feasible solution. The column "Random-MCS" performs Monte Carlo Simulation (MCS, see Section 3.3) with the simple simulation strategy of the "100k-Random" approach. At each node of the search tree the child with the highest average score based on $1,000$ random simulations ($r = 1,000$) is picked. This already improves the "100k-Random" method but does not yet get close to the optimal value.

The last column, "NNI-MCS" is similar to the Random-MCS method, but it differs in a way that the simulation strategy follows the NNI method in a proportional way. So instead of always selecting the nearest customer (in Algorithm 1 done by the *nearestUnvisitedCustomer()* function), the chance p_{v_i} of selecting customer v_i next is equal to

$$p_{v_i} = 1 - \frac{f(v_i)^\alpha}{\bar{n}^\alpha}$$

where $f(v_i)$ $(1 \geq f(v_i) \geq \bar{n})$ returns the current rank of that customer in the list of nearest neighbors. So the nearest neighbor has rank 1, the next one rank 2, etc. Furthermore, $\bar{n}$ is the number of remaining unvisited customers, whereas α could be used to determine the focus more on the nearest solutions. A value of 1 for α turns out to work just fine, as $\bar{n}$ is often a quite large number. The NNI-MCS clearly improves upon all previously discussed methods, and even though the difference with the optimal solution is still quite back, at least all solutions are feasible.

Not much further research was done on the NNI method, because this method is clearly outperformed by the CWS method, which is discussed next. The results in this section do show that Monte Carlo Techniques can actually be useful for the NNI method for two reasons. Firstly because feasible solutions are obtained for all instances while the separate methods (MCS and NNI) could not, and secondly because they improve the quality (minimization of the solution length) of the obtained solutions.

4.4 MCT applied to the CWS method

Clarke & Wright's Savings (CWS) algorithm (see Section 2.3.2) is a savings-based algorithm that produces a list of pairs of customers sorted in descending order by the cost that would be saved when those two customers would be served after each other by the same vehicle, instead of by a separate vehicle. We have tested the CWS method on the test set that we also used for the NNI method in the previous section. The results are outlined in Table 4.2. In this table the first three columns again represent the name of the test instance, the tightness and the optimal value or best known value, respectively. The next column lists the results results of the plain CWS method. As one can see, problems with capacity-infeasible solutions can still occur, though less frequently than with the NNI method. The brackets mean that too many vehicles are used. For the feasible solutions, we do observe that the obtained costs are a lot closer to the optimal value. From this we conclude that the CWS method is likely to be a better heuristic than the NNI algorithm. We have applied Monte Carlo Simulation (MCS, Section 3.3) to the CWS method in two ways, as described in the next two subsections.

Instance	Tight-ness	Best	CWS	BestX-CWS-MCS	Binary-Sampling	Binary-CWS-MCS
`E-n13-k4`	0.76	247	(287)	247 (0%)	247 (0%)	247 (0%)
`E-n22-k4`	0.94	375	388	377 (0.53%)	380 (1.33%)	375 (0%)
`E-n23-k3`	0.75	569	(645)	631 (10.9%)	652 (14.59%)	621 (9.14%)
`E-n30-k3`	0.94	534	(611)	627 (17.42%)	555 (3.93%)	543 (1.69%)
`E-n31-k7`	0.92	379	610	470 (24.01%)	473 (24.80%)	454 (19.79%)
`E-n33-k4`	0.92	835	904	899 (7.66%)	881 (5.51%)	836 (0.12%)
`E-n51-k5`	0.97	521	595	581 (11.52%)	586 (12.48%)	531 (1.92%)
`E-n76-k7`	0.89	682	747	747 (9.53%)	737 (8.06%)	701 (2.79%)
`E-n76-k8`	0.95	$\leq$ 735	817	808 (9.93%)	804 (9.39%)	761 (3.54%)
`E-n76-k10`	0.97	830	(893)	971 (16.99%)	891 (7.35%)	858 (3.37%)
`E-n76-k14`	0.97	1021	(1157)	1201 (17.63%)	1194 (16.94%)	1046 (2.45%)
`E-n101-k8`	0.91	$\leq$ 815	955	950 (16.56%)	908 (11.41%)	867 (6.38%)
`E-n101-k14`	0.93	$\leq$ 1071	1122	1122 (4.76%)	1119 (4.48%)	1098 (2.52%)
`G-n262-k25`	0.97	$\leq$ 6119	(6299)	7006 (14.5%)	6211 (1.50%)	6123 (0.07%)
Total		**14733**		**16637 (12.92%)**	**15638 (6,14%)**	**15061 (2.23%)**

Table 4.2: Solution lengths for various CWS methods.

4.4.1 BestX-CWS

This method is very similar to the methods presented in [6] and [13]. The random simulations in this Monte Carlo Simulation algorithm follow Clarke & Wright's Savings list, and repeatedly select a savings pair s_{ij} for processing from the savings list with probability a p_{ij} that is proportional tot the total savings in the savings list:

$$p_{ij} = \frac{s_{ij}^{\alpha}}{\sum_{k,l} s_{kl}{}^{\alpha}}$$

k and l are the indexes of the unvisited customers. α again defines the focus on the best savings, and can be set to an integer value somewhere between 1 and 5, according to [19]. In our experiments a value of 1 again turned out to work just fine. We performed experiments with a Monte Carlo Simulation algorithm with $r = 1000$ random probes, and correction for each critical misdecision until the maximum execution time of 5 minutes runs out. The experiments suggest that adding more time does not significantly improve the solution quality.

The results are outlined in the "BestX-CWS-MCS" column of Table 4.2. The method always obtains a result equal to or smaller than the solution from the "CWS method". The average deviation percentage is 12.92%, and feasible solutions are found for all test instances. We can conclude that this method always produces a feasible solution, and it produces reasonable results, at least outperforming the CWS method, in terms of solution distance.

4.4.2 Binary-CWS

This method is again based on Monte Carlo Simulation and makes use of Clarke & Wright's Savings (CWS) list as presented in Section 2.3.2. In each simulation of the algorithm, the savings list, sorted in descending order by the size of the savings, is processed linearly from top to bottom. However, in the simulation of the Binary-CWS a savings pair is only processed with a certain probability p ($0 < p < 1$). This means that occasionally, a savings pair is skipped. The question is how to set the value of p. Larger values for p will result in too much chaos, we will still want to process the pairs of "big savers" with high probability (exploitation) to keep the total solution length as low as possible. But then again, if we do not allow enough deviation and set p to a value very close to 1, we will not introduce enough of the exploration element of a Monte Carlo algorithm. We therefore experimented with several values of p on a particular instance, `E-n51-k5`. The result is shown in Figure 4.1. Observe how a large value of p gives a solution equal to that of the CWS method. The optimal value lies somewhere between 0.05 and 0.4. Therefore, in each simulation of our algorithm, p is set to a random value between these two bounds, so $0.05 \leq p \leq 0.4$.

The results of the best out of $100,000$ random simulations based on the technique described above is given in the sixth column of Table 4.2, "Binary-CWS-Sampling". We immediately notice that a feasible solution is again always found for all of the instances, and the result is already a lot closer to the optimal value, for the entire test set the difference is only $15638 - 14733 = 905$, which is a deviation of only 6.14%.

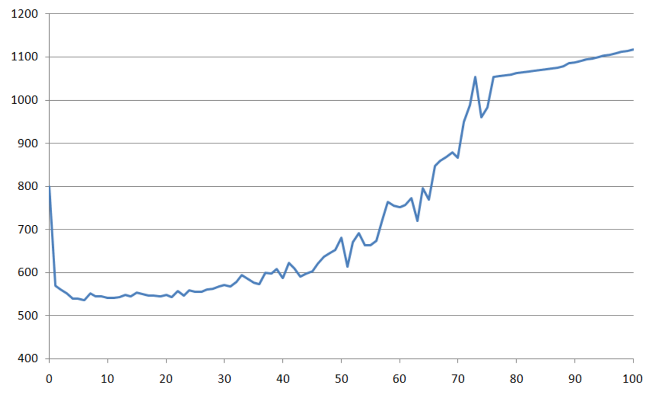


Figure 4.1: Values of p (verticle axis) and their corresponding best found solution (horizontal axis) for E-n51-k5.

Next we used the simulation scheme we just described in a Monte Carlo Simulation algorithm. The search tree is now a binary tree that at each node makes a choice of whether or not a certain savings pair is processed or not. The choice is based on 1,000 simulations that are performed for both of the choices. The branch with the highest average score is chosen next. The method is assisted by a mechanism for the correction of critical misdecisions as described in Section 3.3 for each decision in a certain derivation, until the maximum execution time of 5 minutes runs out. The results are outlined in the rightmost column of Table 4.2, "Binary-CWS-MCS". Notice how using MCS performs almost three times better than using sampling, and reduces the average deviation from the optimal value from 6.14% to just 2.28%. See Figure 4.2 for a visualization of the solution generated by our algorithm for `E-n22-k4`.

To test our method even further, we also tested it on some additional test instances, of which the results are shown in Table 4.3. The reason for performing these additional tests is that this larger test set has more diversity in the number of vehicles and customers and the relation between the two. Also, the `M`-instances have a large amount of customers and vehicles, allowing us to test the scalability of our method. The algorithm clearly also performs well for larger test sets with deviations between 0.22% and 3.61% for instances with 200 customers and 16 or 17 vehicles. See Figure 4.3 as an example of the solution generated by our algorithm for `M-n200-k17`, a large instance that only deviated 0.22% from the best found solution. So again, a deviation of only 2.56% from the optimal or best known value is observed over the entire test set, in line with the previous results on the `E-`-instances.

4.4.3 ALGACEA-2 vs. Binary-CWS-MCS

To our best knowledge, ALGACEA-2 as described in [20] is currently the best Monte Carlo algorithm for the VRP. We tested our Binary-CWS-MCS method on the test set of Symmetric CVRP instances found at [40], the set with which the ALGACEA-2 method was also tested. We used no more than 5 minutes of computation time, which is an amount roughly equal to that of ALGACEA-2. The results are shown in Table 4.4.

Our method produces solutions with a difference of $13046 - 12632 = 414$ distance units (3.28%), whereas ALGACEA-2 has a difference of 1794 (14.20%) from the optimal or best known solution.

Instance	Tightness	Best	CWS	ALGACEA-2	Diff	Binary-CWS-MCS	Diff
A-n65-k9	0.97	1174	1479	1343	14.40%	1224	4.26%
A-n80-k10	0.96	1764	1945	1927	9.24%	1805	2.32%
E051-05E	0.89	525	637	579	10.29%	538	2.48%
E072-04F	0.97	242	345	310	28.10%	265	9.50%
E076-07S	0.97	691	845	781	13.02%	703	1.74%
E076-10E	0.97	837	999	948	13.26%	860	2.75%
E076-14U	0.91	1029	1160	1122	9.04%	1057	2.72%
E101-08E	0.91	826	1031	970	17.43%	861	4.24%
E101-s10C	0.93	820	940	877	6.95%	844	2.93%
E101-14U	0.93	1091	1306	1258	15.31%	1101	0.92%
E151-12C	0.93	1031	1331	1252	21.44%	1084	5.14%
E200-17B	0.94	1291	1291	1557	20.60%	1346	4.26%
E200-17C	0.94	1311	1557	1502	14.57%	1358	3.59%
Total		**12632**	**14866**	**14426**	**14.20%**	**13046**	**3.28%**

Table 4.3: ALGACEA-2 vs. Binary-MCS-CWS.

Interesting to notice is that the instances that are hard for the ALGACEA-2 algorithm are also relatively hard for our Binary-CWS-MCS method. This is most likely due to to the limitation of the applicability of the CWS method to that particular instance.

We think our method performs better because of two reasons. First of all, our method performs Monte Carlo Simulation instead of random sampling. Second, our method respects the order of the savings list, whereas the ALGACEA-2 method can and often will change the order in which the savings are processed. We suspect this ordering to be crucial for obtaining good results. Another advantage of our method is that it is much simpler than the ALGACEA-2 method.

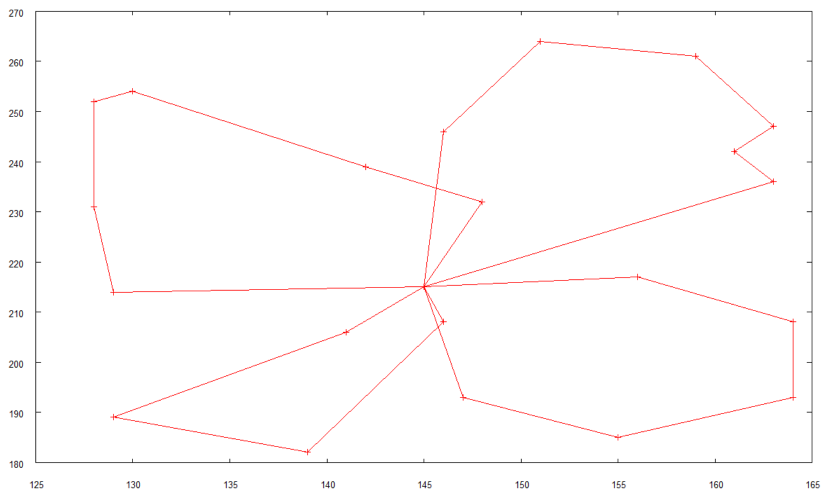


Figure 4.2: Visualization of the solution of instance E-n22-k4 with length 836.

4.5 Discussion

If we look at the results from this chapter we see that the NNI method is really not a good basis for a Monte Carlo algorithm if our goal is to compete with existing (meta-)heuristics.

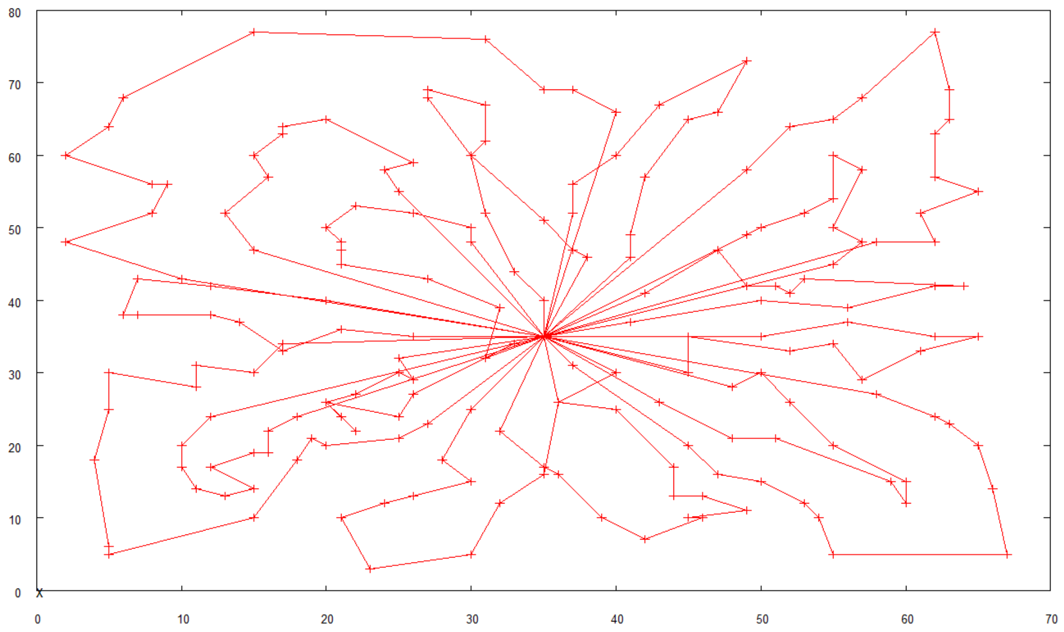


Figure 4.3: Visualization of the solution of instance M-n200-k17 with length 1376.

The CWS method is a much better method, especially when it is assisted by the Monte Carlo Techniques from either the BestX-CWS algorithm or the Binary-CWS algorithm, the technique introduced in this thesis. In essence, we can see the plain CWS method as one path in a very large search space with a branch for each savings pair (only consider the savings pairs applicable to the current built solution). The Monte-Carlo derivatives of the CWS method in essence "walk along" this solution path, making detours from the CWS solution path every now and then with a certain probability. This making of small "detours" from the CWS path performs well, and if we introduce enough diversity (exploration) in these detours, we will ultimately at some point find the optimal solution. But if too much diversity is introduced, the search space will be way too large and again filled with all the bad solutions.

The problem is that the power of these methods is limited by the performance of the CWS method. For some instances, the CWS method is simply not the way to go, and no matter how much we deviate from the CWS solution path, or no matter how many and which savings pairs we skip, we never find that one optimal solution. We can conclude that the Monte Carlo methods we described in this section are limited by the applicability of the methods that they are based on. Nevertheless, the results are good, and on average not more than a few percent away from the optimal or best known solutions.

Instance	Tightness	Best	Binary-CWS-MCS	Diff
A-n32-k5	0.82	784	853	8.80%
A-n33-k5	0.89	661	661	0.00%
A-n33-k6	0.90	742	758	2.16%
A-n34-k5	0.92	778	794	2.06%
A-n36-k5	0.88	799	805	0.75%
A-n37-k5	0.81	669	679	1.49%
A-n37-k6	0.95	949	968	2.00%
A-n38-k5	0.96	730	786	7.67%
A-n39-k5	0.95	822	823	0.12%
A-n39-k6	0.88	831	841	1.20%
A-n44-k6	0.95	937	947	1.07%
A-n45-k6	0.99	944	992	5.08%
A-n45-k7	0.91	1146	1160	1.22%
A-n46-k7	0.86	914	919	0.55%
A-n48-k7	0.89	1073	1103	2.80%
A-n53-k7	0.95	1010	1057	4.65%
A-n54-k7	0.96	1167	1174	0.60%
A-n55-k9	0.93	1073	1096	2.14%
A-n60-k9	0.92	1354	1366	0.89%
A-n61-k9	0.98	1034	1069	3.38%
A-n62-k8	0.92	1288	1312	1.86%
A-n63-k9	0.97	1616	1646	1.86%
A-n63-k10	0.93	1314	1351	2.82%
A-n64-k9	0.94	1401	1430	2.07%
A-n65-k9	0.97	1174	1224	4.26%
A-n69-k9	0.94	1159	1224	5.61%
A-n80-k10	0.94	1763	1805	2.38%
M-n151-k12	0.93	1053	1090	3.51%
M-n200-k16	1.00	1470	1523	3.61%
M-n200-k17	0.94	1373	1376	0.22%
Total		**32028**	**32832**	**2.56%**

Table 4.4: Solution lengths for Binary-MCS-CWS.

Chapter 5

Conclusion

The Vehicle Routing Problem (VRP) has been studied for over fifty years. Many variants have been considered, all with roots in different popular areas of computer science (graph theory, bin packing, scheduling, stochastic environments). The VRP is and remains interesting in two ways: it serves as an excellent testing platform for new search methods and (meta-)heuristics, while at the same time solving the actual problem more optimally does not only have scientific value but will also always remain beneficial for the industry sector that profits greatly from reducing transportation costs.

Up until today, a good exact method for solving the Vehicle Routing Problem as we see it appear in practice has not yet been found for general instances with more than say 60 vehicles (which is not at all an unusual amount in real-life versions of this problem). We thus have to rely on heuristic methods that give an as optimal as possible solution. Nearest Neighbor Insertion, Giant Tour Based Algorithms and Clarke & Wright's Savings (CWS) algorithm have been suggested as heuristic methods, of which the latter is especially interesting as input or guidance for other algorithms. Ant Colony Algorithms, Tabu Search/Simulated Annealing and Genetic algorithms are all examples of proven metaheuristics for the VRP that currently produce the most optimal solutions.

Monte Carlo Techniques can be a useful method to traverse search trees where it is hard to derive an admissible heuristic as guidance for branch-and-bound or other heuristic best-first-search methods. We have applied Monte Carlo Techniques to the NNI method to illustrate how these techniques can indeed improve existing methods and make them produce feasible results. We then turned our attention to the CWS algorithm, tested some existing methods, and developed a new method called Binary-CWS-MCS which is a Monte Carlo Simulation algorithm. The key lies in the simulation strategy, that processes the savings list from the CWS algorithm in descending order just like the original algorithm, but skips a savings pair with a certain probability. This method produces solutions for instances out of popular test sets with only a 3.28% deviation from the optimal solutions, outperforming the other Monte Carlo based methods from literature.

We end this conclusion with a more general remark. In complex environments with associated NP-hard (or NP-complete) decision or optimization problems (such as the Vehicle Routing Problem) and no place for exact algorithms, heuristics must be used. These heuristic methods often share a common property: they all somehow have to deal with the trade off between exploration and exploitation. Finding the right balance between the two is and remains one of the most interesting and hardest tasks for any heuristic algorithm. Optimizing this trade-off is in essence the biggest problem behind every single algorithm described in this thesis.

Future Work

In our case we have tried to apply Monte Carlo Techniques to various existing solving methods for the Vehicle Routing Problem. There are many more solving methods for the VRP suggested in literature which may also benefit from Monte Carlo Techniques, for example the various Tour Splitting Algorithms. Some work can still be done on our method, Binary-CWS-MCS. It can most likely be improved by fine-tuning the value of p, for example by making it dynamic and dependent on some domain-specific property of the VRP (though so far we have been unable to link it to the right property). Some additional local TSP optimization could also be done to improve the obtained solution quality. The Binary-CWS simulation strategy can perhaps also be used in the MCTS algorithm, if a suitable backpropagation strategy can be found. And if one is not interested in improving existing methods, why not try and find yet another solving method for the VRP that can compete with, or even outperform the existing VRP solving methods?

Bibliography

[1] C. Archetti, M. Savelsbergh, and M. Speranza, *Worst-case analysis for split delivery Vehicle Routing Problems*, Transportation Science, 40 (2006), pp. 226–234.

[2] B. Baker and M. Ayechew, *A genetic algorithm for the Vehicle Routing Problem*, Computational Operational Research, 30 (2003), pp. 787–800.

[3] R. Baldacci and A. Mingozzi, *Lower bounds and an exact method for the Capacitated Vehicle Routing Problem*, Service Systems and Service Management, 2 (2006), pp. 1536–1540.

[4] J. C. Beck, P. Prosser, and E. Selensky, *Vehicle routing and job shop scheduling: What's the difference?*, in Proceedings of the 13th International Conference on Artificial Intelligence Planning and Scheduling, 2004, pp. 267–276.

[5] L. Bodin, B. Golden, A. Assad, and M. Ball, *Routing and scheduling of vehicles and crews*, The State of the Art: Computers and Operations Research, 10 (1986), pp. 63–212.

[6] G. Buxey, *The Vehicle Scheduling Problem and Monte Carlo Simulation*, Journal of Operational Research Society, 30 (1979), pp. 563–573.

[7] T. Cazenave, *Nested Monte-Carlo search*, in IJCAI'09: Proceedings of the 21st International Joint Conference on Artifical intelligence, Morgan Kaufmann Publishers Inc., 2009, pp. 456–461.

[8] N. Christofides, A. Mingozzi, and P. Toth, *Exact algorithms for the Vehicle Routing Problem, based on spanning tree and shortest path relaxations*, Mathematical Programming, 20 (1981), pp. 255–282.

[9] G. Clarke and J. Wright, *Scheduling of vehicles from a central depot to a number of delivering points*, Operations Research, 12 (1964), pp. 568–581.

[10] J. Cordeau and G. Laporte, *Tabu search heuristics for the Vehicle Routing Problem*, in Operations Research/Computer Science Interfaces, Springer US, 2005, pp. 145–163.

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

[12] P. de Córdoba, L. García-Raffi, and J. Sanchis, *A heuristic algorithm based on Monte Carlo methods for the rural postman problem*, Computers and Operations Research, 25 (1998), pp. 1097–1106.

[13] P. F. de Córdoba, L. García-Raffi, A. Mayado, and J. Sanchis, *A real delivery problem dealt with Monte Carlo Techniques*, Sociedad de Estadistica e Investigacion Operativa Top, 8 (2000), pp. 57–71.

[14] J. Dethloff, *Vehicle routing and reverse logistics: The Vehicle routing problem with simultaneous delivery and pick-up*, OR Spectrum, 23 (2001), pp. 79–96.

[15] M. Dorigo and T. Sttzle, *Ant Colony Optimization*, MIT Press, 2004.

[16] B. Dorronsoro, *The VRP Web*, [accessed November 30, 2009], `http://neo.lcc.uma.es/radi-aeb/WebVRP/`.

[17] M. Dror, *ARC Routing: Theory, Solutions and Applications*, Springer-Verlag New York, 2006.

[18] A. Eiben and J. Smith, *Introduction to Evolutionary Computing*, Springer, 2003.

[19] J. Faulin, M. Gilibert, A. Juan, X. Vilajosana, and R. Ruiz, *SR-1: A simulation-based algorithm for the Capacitated Vehicle Routing Problem*, in Proceedings of the 40th Conference on Winter Simulation, Winter Simulation Conference, 2008, pp. 2708–2716.

[20] J. Faulin and A. Juan, *ALGACEA-2: An entropy-based heuristics for the Capacitated Vehicle Routing Problem*, Springer US, 2007.

[21] J. Faulin and A. Juan, *The ALGACEA-1 method for the Capacitated Vehicle Routing Problem*, International Transactions in Operational Research, 15 (2008), pp. 599–621.

[22] M. L. Fisher, *Optimal solution of Vehicle Routing Problems using minimum k-trees*, Operations Research, 42 (1988), pp. 626–642.

[23] B. Golden, S. Raghavan, and E. Wasil, *A new tabu search heuristic for the site-dependent Vehicle routing problem*, in The Next Wave in Computing, Optimization, and Decision Technologies, Springer US, 2005, pp. 107–119.

[24] R. Hassi and S. Rubinstein, *On the complexity of the k-customer Vehicle Routing Problem*, Operations Research Letters, 33 (2005), pp. 71–76.

[25] L. Kocsis and C. Szepesvari, *Bandit based Monte-Carlo planning*, in Proceedings of the 17th European Conference on Machine Learning, Springer, 2006, pp. 282–293.

[26] G. Laporte and Y. Nobert, *Exact algorithms for the Vehicle Routing Problem*, Annals of Discrete Mathematics, 31 (1987), pp. 147–184.

[27] A. M. Law and W. D. Kelton, *Simulation Modeling and Analysis*, Third edition, McGraw-Hill, 2000.

[28] H. Longo, M. de Arago, and E. Uchoa, *Solving Capacitated Arc Routing Problems using a transformation to the CVRP*, Computers and Operations Research, 33 (2006), pp. 1823–1837.

[29] S. Mazzeo and I. Loiseau, *An ant colony algorithm for the Capacitated Vehicle Routing*, Electronic Notes in Discrete Mathematics, 18 (2004), pp. 181–186.

[30] J. R. Montoya-Torres, E. A. Lizarazo, E. Gutierrez-Franco, and A. X. Halabi, *Using randomization to solve the deterministic single and multiple Vehicle Routing Problem with service time constraints*, in Proceedings of the 2009 Winter Simulation Conference, 2009, pp. 2989–2994.

[31] I. Osman, *Metastrategy Simulated Annealing and Tabu Search Algorithms for Combinatorial Optimization Problems*, Ph.D. Thesis, The Management School, Imperial College, London, 1991.

[32] W. Pearn, A. Assad, and B. Golden, *Transforming Arc Routing into node routing problems*, Computers and Operations Research, 14 (1987), pp. 285–288.

[33] C. Prins, N. Labadi, and M. Reghioui, *Tour splitting algorithms for Vehicle Routing Problems*, International Journal of Production Research, 47 (2009), pp. 507–535.

[34] T. Ralphs, *Branch and Cut.org*, [accessed November 30, 2009], `http://www.branchandcut.org/`.

[35] G. Reinelt, *TSPLIB: A Traveling Salesman Problem library*, ORSA Journal on Computing, 3 (1991), pp. 376–384.

[36] S. Russell and P. Norvig, *Artificial Intelligence, A Modern Approach*, Second edition, Pearson Education, 2003.

[37] M. Schadd, M. Winands, H. van den Herik, and H. Aldewereld, *Addressing NP-complete puzzles with Monte-Carlo methods*, in Proceedings of the AISB 2008 Symposium on Logic and the Simulation of Interaction and Reasoning, vol. 9, The Society for the Study of Artificial Intelligence and Simulation of Behaviour, 2008, pp. 55–61.

[38] P. Toth and D. Vigo, eds., *The Vehicle Routing Problem*, SIAM, 2002.

[39] J. van Woensel, L. Kerbache, H. Peremans, and N. Vandaele, *Vehicle routing with dynamic travel times: A queueing approach*, European Journal of Operational Research, 186 (2008), pp. 990–1007.

[40] D. Vigo, *DEIS - Operations Research Group Library of Instances*, [accessed Januari 31, 2010],
`http://www.or.deis.unibo.it/research_pages/ORinstances/VRPLIB/VRPLIB.html`.

[41] Wikipedia, *Vehicle Routing Problem*, [accessed November 30, 2009],
`http://en.wikipedia.org/wiki/{V}ehicle_routing_problem`.

[42] S. Wohlk, *A decade of Capacitated Arc Routing*, in The Vehicle Routing Problem: Latest Advances and New Challenges, Springer Science, 2008, pp. 29–48.

List of Figures

List of Tables

Appendix

Below is an overview of the obtained solutions using the Binary-CWS-MCS method described in Chapter 4.

```
E-n13-k4:
Route #1: 3 5 8
Route #2: 11 4 7 2
Route #3: 6 10 12 9
Route #4: 1
Cost 247

E-n23-k3:
Route #1: 6 1 2 5 7 9
Route #2: 10 8 3 4 11 13
Route #3: 16 19 21 14
Route #4: 12 15 18 20 17
Cost 375

E-n30-k3:
Route #1: 21 6 24 25 29 27 28 26
Route #2: 20 3 4 1 5 2 22
Route #3: 19 10 12 14 8 9 17 7 13 16 15 11 23 18
Cost 543

E-n31-k7:
Route #1: 24 3 14 5 6 22 8 2
Route #2: 20 7 12 26
Route #3: 23 29
Route #4: 30
Route #5: 25 21 18 10
Route #6: 16 11 13 9 28 15
Route #7: 19 1 17 4 27
Cost 454

E-n33-k4:
Route #1: 5 6 9 10 18 19 22 21 20 23 24 25 17 13
Route #2: 1 15 26 27 16 28 29
Route #3: 4 7 8 32 11 12 2 3
Route #4: 31 14 30
Cost 836

E-n51-k5:
Route #1: 1 22 20 35 36 3 28 31 26 8 27
Route #2: 4 17 44 42 19 40 41 13 25 14
Route #3: 12 37 15 45 33 39 30 10 49 5
Route #4: 32 2 16 29 21 34 50 9 38 11 46
Route #5: 6 48 23 7 43 24 18 47
Cost 531

E-n76-k7:
Route #1: 75 30 48 47 36 69 71 60 70 20 37 5 29 45
Route #2: 51 16 63 23 56 41 42 64 43 1 73 33 6
Route #3: 12 72 39 9 25 55 31 10 58 26
Route #4: 7 11 38 65 66 59 14 53 35
Route #5: 17 3 49 24 18 50 32 44 40
Route #6: 68 2 62 28 22 61 21 74 4
Route #7: 67 34 52 27 15 57 13 54 19 8 46
Cost 701

E-n76-k8:
Route #1: 30 48 47 36 69 71 60 70 20 37 5 29
Route #2: 51 33 63 23 56 41 42 64 43 1 73
Route #3: 17 12 72 39 9 25 55 31 10 58
Route #4: 26 38 65 66 11 7 67
Route #5: 16 49 24 18 50 44 3 32 40
Route #6: 35 53 14 59 19 8 46 34
Route #7: 68 2 62 28 22 61 21 74 6
Route #8: 75 4 45 27 15 57 13 54 52
Cost 761

E-n76-k10:
Route #1: 48 47 36 71 60 70 20 37 5
Route #2: 73 1 43 41 42 64 22 62 68
Route #3: 72 10 31 25 55 18 50 32
Route #4: 38 65 66 59 14 19
Route #5: 2 28 61 69 21 74 30
Route #6: 6 33 63 23 56 24 49 16
Route #7: 75 4 45 29 15 57 27
Route #8: 26 12 58 11 53 35 7
Route #9: 51 44 9 39 40 3 17
Route #10: 67 34 52 13 54 8 46
Cost 858

E-n76-k14:
Route #1: 36 69 71 60 70 20 37 27
Route #2: 43 41 42 64 22 61
Route #3: 25 55 31 10 58
Route #4: 38 65 66 59
Route #5: 49 24 18 50 32
Route #6: 29 5 15 57 13 54 19
Route #7: 74 21 47 48 30
Route #8: 11 14 53 35
Route #9: 40 9 39 72 12
Route #10: 68 6 3 44 17 26
Route #11: 33 62 28 2
Route #12: 7 8 46 52 45
Route #13: 75 4 34 67
Route #14: 51 16 63 23 56 1 73
Cost 1046

E-n101-k8:
Route #1: 76 77 3 79 78 34 35 65 71 66 20 30 1 69
Route #2: 84 17 45 46 47 36 49 64 11 19 48 82 8 83
Route #3: 21 73 72 56 23 39 67 25 55 4 54 26
Route #4: 87 42 43 14 38 86 44 16 61 5 60 18
Route #5: 52 7 88 62 10 63 90 32 70 31 27
Route #6: 50 33 81 9 51 24 29 80 68 12 28
Route #7: 53 58 40 74 75 22 41 15 57 2 13 94
Route #8: 89 6 96 99 59 93 85 91 100 98 37 92 97 95
Cost 867

E-n101-k14:
Route #1: 3 78 34 35 65 71 66 20 70
Route #2: 52 7 19 49 64 11 62 88
Route #3: 18 8 46 36 47 48 82
Route #4: 39 67 23 56 75
Route #5: 92 91 44 14 38 86 16
Route #6: 31 10 63 90 32 30
Route #7: 26 4 25 55 54 12 28
Route #8: 58 2 57 15 43 42 87 97 95
Route #9: 68 80 24 29 79 77 76
Route #10: 21 72 74 22 41 73 40 53
Route #11: 60 5 84 17 45 83 89
Route #12: 93 61 85 100 98 37
Route #13: 13 94 59 99 96 6
Route #14: 50 33 81 9 51 1 69 27
Cost 1098

G-n262-k25:
Route #1: 91 50 88 151 86 149 94 247 77 212
Route #2: 164 259 35 239 119 218 184 192 144 156
Route #3: 165 257 253 117 98 240 107 14 143 174 48 23 118 123
Route #4: 261 79 71 1 229 31 205 221 129 130 49 124 255
Route #5: 207 139 202 231 232 142 68 234 33 72 166
Route #6: 128 179 161 45 187 93 46 216 63 250 126
Route #7: 135 84 181 171 132 41 210 59 162 75 219
Route #8: 67 51 134 159 147 245 201 78 32 226 233 122
Route #9: 158 60 110 109 20 227 29 97
Route #10: 140 172 145 115 154 87 76 155 108 70 99 252
Route #11: 74 7 160 28 182 34 95 188 256 4 81 186 213
Route #12: 6 222 85 92 178 105 169 146 62
Route #13: 254 66 168 47 175 55 17 40 127 24 258 136 176
Route #14: 215 104 141 25 246 208 152 180 200 199 116
Route #15: 237 3 37 111 196 236 120 22 44 8
Route #16: 18 12 61 90 211 133 5 191 30
Route #17: 131 238 10 42 197 194 103 113 80 56 242
Route #18: 195 58 189 106 121 248 9 26 148 114 101
Route #19: 204 54 137 53 228 173 260 27 125 82 163 69
Route #20: 217 170 2 230 153 89 167 13 251
Route #21: 36 183 150 220 65 206 83
Route #22: 96 19 138 209 244 100 21 185
Route #23: 249 157 43 225 241 11 214 57 203 235
Route #24: 16 243 52 15 102 39 38 223 193
Route #25: 64 224 177 190 73 198 112
Cost 6123
```