Applying Monte Carlo Techniques to the Capacitated Vehicle Routing Problem

Frank W. Takes Walter A. Kosters

Leiden Institute of Advanced Computer Science, Leiden University, The Netherlands

Abstract

This paper describes a new method for solving the Capacitated Vehicle Routing Problem (CVRP). The CVRP is defined as the problem of delivering goods from a depot to a number of customers with a fleet of capacitated vehicles, minimizing the cost of distributing the goods while respecting the capacity constraints of the vehicles. We have developed a method based on Monte Carlo Simulation (MCS) and the Clarke & Wright Savings (CWS) heuristic that, with similar memory and CPU usage, outperforms previous best known Monte-Carlo-based algorithms for the CVRP. Performance is comparable with that of the current state-of-the-art methods based on metaheuristics.

1 Introduction

The *Vehicle Routing Problem (VRP)* is a widely studied [17] NP-hard [16] combinatorial optimization problem that was introduced in 1959 by Dantzig and Ramser [9]. In the capacitated variant we are given a number of customers (nodes), each with its own numerical demand, the distance between each pair of customers (edges), and a number of capacitated vehicles that can, starting from the depot (a special node), traverse the edges and serve the customers. Many variants of the problem have been suggested, but the main objective is to minimize the total distance traveled while respecting the capacity constraints of the vehicles.

The problem is not only of scientific value, but it is also important to the industry sector. 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.

The term *Monte Carlo Techniques (MCT)* is used to describe a class of widely used algorithms that rely on the use of random sampling to finally acquire a solution to an optimization problem. Often, the decisions made when traversing the search space are based on performing random simulations for the possible successor states. We will see that these techniques can also be applied to the VRP.

Section 2 starts with a description of the Vehicle Routing Problem and its variants. Several solution methods are presented in Section 3. Monte Carlo Techniques are introduced in Section 4, and Section 5 describes how these techniques can be applied to the Vehicle Routing Problem. We will propose a new Monte-Carlo-based algorithm and study its performance in Section 6. Section 7 concludes.

2 Vehicle Routing Problem

This section outlines and formally defines the VRP, ending with an overview of the most common variants.

2.1 Problem Definition

A *customer* is an entity that has a certain *demand* and therefore requires the presence of a *vehicle*, a "salesman" 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 for a certain vehicle, and starts and ends at the depot. The *goal* of the basic VRP is to serve all customers, minimizing the total cost of the routes of all vehicles. Depending on the considered variant of the VRP (see Section 2.3), additional constraints, such as vehicle capacities, may apply. A visual example of the VRP is given in Figure 1.

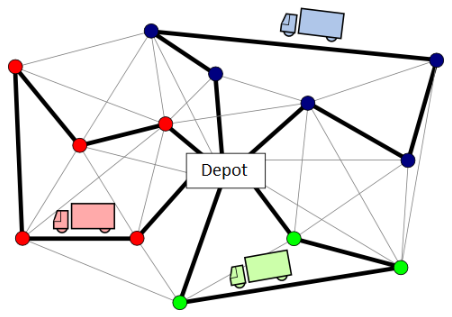

Figure 1: Example of a VRP instance with $n = 13$ and $m = 3$. To keep visualization simple, the graph is not complete. Image courtesy of Markus Chimani, Univ. Lena

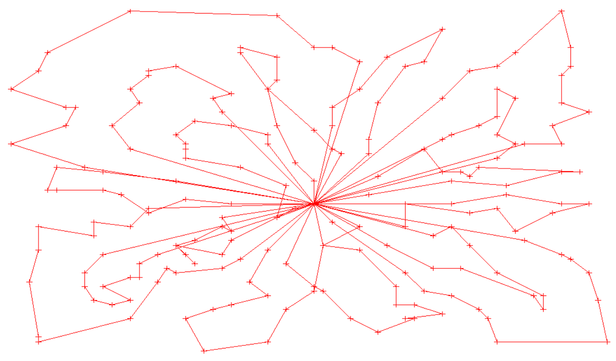

Figure 2: Visualization of a solution of an instance with $n = 200$ and $m = 17$ (instance `200-17B`, see Table 2)

2.2 Formal Definition

The underlying structure of the VRP is a complete (fully connected) directed graph $G(V, E)$. This structure and notations with respect to vehicles, routes and the associated cost is outlined in the following definitions:

- $V = \{v_0, v_1, \ldots, v_n\}$ is a set of $n+1$ ($n \geq 1$) vertices. We distinguish the depot v_0 and n customers.
- $E = \{(v_i, v_j) \mid 0 \leq i, j \leq n, i \neq j\}$ is the set of edges (arcs) between the vertices, called the *roads*.
- $C = (c_{ij})$ is a cost matrix, with $c_{ij} > 0$ representing the cost of traversing edge (v_i, v_j). For all i, we define $c_{ii} = 0$. The triangle inequality holds: $c_{ij} \leq c_{ik} + c_{kj}$ ($0 \leq i, j, k \leq n$). If $c_{ij} = c_{ji}$ for all customers i and j, we talk about the *Symmetric* VRP. If there exist customers i and j for which $c_{ij} \neq c_{ji}$, we consider it to be an *Asymmetric* VRP. We will also denote $c_{ij} = c(v_i, v_j)$.
- m, with $1 \leq m \leq n$, 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 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 to keep notation simple.
- $d = (d_0, d_1, \ldots, d_n)$ with all $d_i > 0$ ($1 \leq i \leq n$) is a vector of the customer *demands*; d_0, the demand of the depot, is always equal to 0. We will also denote the demand d_i of customer v_i as $d(v_i)$.

We have now defined the Vehicle Routing Problem in terms of a minimization problem (or actually, as a Multiple Traveling Salesman Problem), as the main task at hand is to minimize the value of $C(S)$. The constraint that is related to the correlation between the demands and the vehicle routes is specific to the considered variant of the VRP, and therefore outlined in the next subsection.

2.3 Variants

This subsection describes the most common variants of the Vehicle Routing Problem that have been suggested in literature. 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.

The *Capacitated Vehicle Routing Problem (CVRP)* is the most common variant of the Vehicle Routing Problem. In this variant, a fixed fleet of m 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. The CVRP has a homogeneous and heterogeneous variant.

In the *Homogeneous* CVRP (or *Uniform Fleet* CVRP) each vehicle 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_i does not exceed the vehicle capacity Q: $\sum_{j=1}^{k_i} d(v_j^i) \leq Q$. To ensure that vehicles are always big enough, the demand of a customer is never greater than the vehicle capacity: $d_j \leq Q$ $(1 \leq j \leq n)$. Also, the total demand of all customers can not be greater than the total capacity of all vehicles: $\sum_{j=1}^{n} d_j \leq m{*}Q$.

In the *Heterogeneous* CVRP (or *Mixed Fleet* CVRP) the fleet is composed of different vehicle types, each with its own capacity, and sometimes also with its own fixed cost. Even the cost matrix C can be specific for each vehicle. A more detailed analysis of this variant is given in [22].

In the Vehicle Routing Problem *with Time Windows* each customer has to be served within a specific time window. Adding these windows to the VRP creates a link with the well-known *Job Scheduling Problem*, see [3]. In the VRP *with Pickup and Delivery* customers may also return items; this variant is outlined in [11]. The *Stochastic* VRP covers all the variants of the VRP with one or more randomized properties. For example, the variant where the distance matrix is dynamic is described in [24]. Any variant that has a certain bound D on the maximum distance covered by a vehicle is called a *Distance Constrained* VRP. Time may also be dependent on other properties than distance, and when a maximum allowed time T per vehicle or for the entire routing is specified, we consider the VRP to be *Time Constrained*.

This paper will focus on the Symmetric Homogeneous Capacitated Vehicle Routing Problem, in the sequel addressed as "*the CVRP*", or even shorter for reader convenience, "*the VRP*". This VRP variant has been proven to be NP-hard. For a detailed complexity analysis we refer the reader to [16]. We remark that the hardness of an instance of the VRP is often related to the *tightness T*, which is the relation between the sum of the demands and the total capacity of all the vehicles: $T = \sum_{i=1}^{n} d_i \ / \ (Q * m)$.

3 Related Work

Many solving methods for the CVRP have been proposed. We distinguish between *exact methods*, methods based on *heuristics*, and those based on *meta-heuristics*. In this section, we will describe each of these approaches, paying special attention to one heuristic method, which will be the basis of our algorithm.

3.1 Exact Methods

The easiest exact method that one can think of is a simple *brute-force* approach. In essence we would be 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 $n-1$ customers (placing it after the last customer creates an empty vehicle), creating a total of $n!\binom{n-1}{m-1}/m!$ possible solutions (we divide by $m!$ because the order of the vehicles is irrelevant). Because of this extremely large number of possibilities, we can conclude that it is unlikely that we will ever find the optimal solution with a brute-force approach.

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 [2]. Pure branch-and-bound is still rather slow and will not give good solutions when the value of n increases. A description of a *branch-and-cut* algorithm for the VRP is given in [15]. The algorithm converts the graph behind the VRP into a so-called "K-tree", a structure for which a polynomial algorithm exists to find shortest paths. Edges between certain clustered customers are also fixed, and 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, and works well for problems with up to 100 customers.

Because exact methods are rather limited in performance, we refer the reader to the excellent surveys in [18] and [22] for an overview of other exact algorithms for the VRP.

3.2 Heuristic Methods

Heuristic methods are a class of methods that do not provide an exact answer, but instead produce an as optimal as possible solution in a reasonable amount of time. In this section we describe the Clarke & Wright's Savings heuristic algorithm in detail. Many other algorithms have been developed, see [22] for an overview.

The *Clarke & Wright's Savings (CWS)* algorithm dates back to 1964 when it was introduced in [6] as the first *savings-based algorithm* (sometimes also referred 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

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

(a) Distance matrix

v_i	d_i
v_1	37
v_2	35
v_3	30
v_4	25
v_5	32

(b) Demand vector

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

(c) Savings

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

(d) Savings List

Table 1: Example VRP instance with $n = 5$ for the CWS method

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 processing the so-called *savings list*, which is a list of all customer pairs sorted in descending order by their savings value. 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:

$$s_{ij} = c_{0i} + c_{0j} - c_{ij} \tag{1}$$

Due to the triangle inequality $s_{ij} \geq 0$ for all customers i, j. When the savings list has been processed, two things can happen. Either all customers have been combined into m or less routes and the instance is solved, or more than m routes are still present, as some customers are still assigned to their own vehicle. This problem with so-called *capacity-infeasible* solutions being generated is often caused by a high tightness value. The CWS method thus does not guarantee a capacity-feasible solution, however, when a solution is found, it is likely to be good, as a more or less greedy algorithm to maximize savings has been performed.

Example. Consider the symmetric distance matrix in Table 1a for $n = 5$ customers and the demand vector given in Table 1b. Assume that we have $m = 2$ vehicles, each with capacity $Q = 100$. 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, resulting in Table 1c. For convenience we sort the customer pairs by savings, in descending order, creating the savings list (Table 1d).
We will describe how the CWS algorithm would process the savings list. First, $1-5$ is processed and inserted in the first route. $1-2$ is then skipped because adding 2 would exceed the capacity constraint of our vehicle ($37 + 35 + 32 > 100$). Next $2-4$ is inserted in a new route. Only customer 3 is left now and is first encountered in the $3-5$ pair, so it is added to the first route. In this way the algorithm constructs the routes $1-5-3$ and $2-4$ with a total cost of 171, which in this case also happens to be optimal.

The CWS algorithm has a complexity of $O(n^2 \cdot \log\ n)$ with $n \geq 1$ customers, assuming the savings list is implemented as a heap with n^2 elements, where extraction from the heap takes $\log\ n$ time. The method has very often been adjusted, improved and tuned. As we will see later on, the CWS algorithm can serve as a good basis for other (meta-heuristic) algorithms.

3.3 Meta-heuristics

Meta-heuristics can roughly be categorized into *local search*, *population search* and *learning algorithms*. Each of these meta-heuristics have been applied more than once to the VRP. *Tabu search* is the most popular local search method that has very often been successfully applied [7]. *Genetic algorithms* are also very useful in solving the VRP; these population search techniques are discussed in great detail in [1]. For the VRP, *Ant Colony Optimization (ACO)* [20] is the most frequently applied learning algorithm. Currently, the best algorithms for larger VRP instances are based on Tabu Search and Genetic Algorithms.

4 Monte Carlo Techniques

Monte Carlo Techniques (MCT) are a class of algorithms that rely on the use of random sampling to finally acquire a solution to a given problem. *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. The process 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 plain random sampling compared to for example Depth-First-Search (DFS), is that diversity is maximized. However, the disadvantage is that good solutions are not always randomly distributed over the search tree, but even though scattered, are clustered in some part of the tree. It would

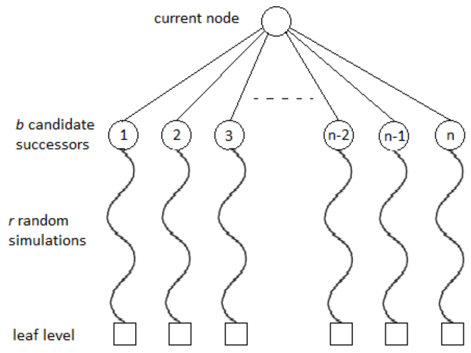

Figure 3: Basic Monte Carlo Simulation

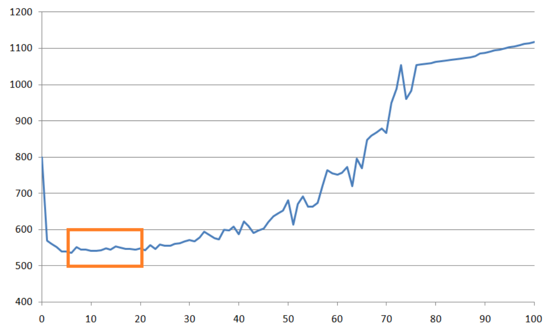

Figure 4: Values of p ($\times 100\%$, horizontal axis) and their corresponding average solution lengths (vertical axis)

be better if we could guide the search towards these better clusters. The key lies in the balance between *exploration* and *exploitation*: the algorithm needs a sufficient amount of randomness to explore the search space, but it also has to exploit already found potentially better regions of the search space.

Monte Carlo Simulation (MCS), for example as described in [19], is especially useful when it is hard to judge the quality of a partial solution before the entire solution has been generated, which would be a requirement for heuristic algorithms such as (ID)A*. The process is explained in Figure 3, 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 $b \geq 1$ possible successors (children) of this current node we perform $r \geq 1$ (common values are around 1000 or more) random simulations until a terminal node of the search space is reached. So at each step of this algorithm a total of $b * r$ so-called *probes* are sent down towards the leaf nodes, following some *simulation strategy*. The "best" child based on these random simulations is then selected for expansion. The "best" child can 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 behaviour of the search space. When the current node is a leaf, an actual solution has been found and the method is terminated. The emerging behaviour here is what we are looking for: the search is repeatedly "guided" towards better parts of the search tree where again better solutions can be found. Another advantage of MCS but also Monte Carlo Techniques in general is that these methods are highly parallelizable.

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, at a global level, keeping track of the best solution found so far during all the random probes, as this solution may not always be identical to the solution that is finally found by the MCS algorithm itself. Also, it is often worthwhile to optimize the simulation strategy. Instead of sending random probes down the search tree, domain-specific knowledge can be used to slightly guide the search. For example, in case of the Vehicle Routing Problem, it might not be smart to create a path between two customers with a very large distance in between them, while both customers have many other closer customers that they can be connected to. The problem with having a big random factor playing a role in an algorithm such as MCS 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 as *Meta Search*, the so-called *restarts* can help to improve the quality of the final solution.

5 Monte Carlo Techniques applied to the VRP

Compared to the huge amount of work done on the VRP, relatively little work seems to have been done on applying Monte Carlo Techniques to the Capacitated VRP. Using Monte Carlo Techniques to solve the VRP was suggested for the first time in 1979 in [5]. 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 [10], where a simulation method very similar to the Nearest Neighbor Insertion [4] method was used. Compared with best known methods, performance was quite limited. Other than the algorithm outlined in the next subsection, no other literature than the above, was found on the application of MCT to the CVRP.

5.1 ALGACEA

In 2007, the ALGACEA-1 method was suggested in [14] as a Monte Carlo algorithm for the VRP, based on the Clarke & Wrights Savings approach. Later on, ALGACEA-2, an improved version assisted by an entropy function and some other improvements was introduced in [13]. In essence, this algorithm performs plain random sampling where in each random sample a savings pair s_{ij} is repeatedly selected from the set of savings pairs with probability p_{ij}:

$$p_{ij} = \frac{{s_{ij}}^{\alpha}}{\sum_{k,\ell} {s_{k\ell}}^{\alpha}} \tag{2}$$

Here k and l are the indexes of the unvisited customers and α defines the focus on the best savings pairs, and can according to the authors be set to an integer value somewhere between 1 and 5. This algorithm thus selects savings pairs with a higher savings value with a higher probability than pairs that produce lower savings, in an attempt to keep the balance between exploration and exploitation intact.

SR-1 [12] and SR-2 are are Monte-Carlo algorithms very similar to ALGACEA, but unfortunately only tests on randomly distributed search spaces were reported, and not the well-known test sets.

5.2 BinaryMCS-CWS

In this section we introduce our method, *BinaryMCS-CWS*, which is based on Monte Carlo Simulation (see Section 4) and the Clarke & Wright's (CWS) algorithm (see Section 3.2). The method is outlined in Algorithm 1, and it essentially iterates over the savings list, sorted in descending order by the size of the savings. If a savings pair is feasible, r random simulations are performed for the current state and the state where that savings pair is processed. If the average solution quality is better when the pair is processed, it is actually processed and the procedure is repeated, otherwise the savings pair is skipped. The algorithm ends when a solution is found or when the savings list has been exhausted. The main difference with the ALGACEA algorithm lies in the fact that our algorithm never deviates from the order of the savings list.

In each random simulation (Algorithm 2), the savings list is processed linearly from top to bottom. However, a savings pair is either skipped with a certain probability p ($0 \leq p \leq 1$) or processed with probability $1 - p$. Notice that setting this parameter p to 0 is the exact CWS algorithm: no savings pairs are skipped and every simulation will have the same outcome. Very high values for p will in turn result in a lot of chaos and infeasible (or at least inefficient) routes. In essence, this probability parameter p allows us to tune the balance between exploration and exploitation: it defines how much diversity is added to the CWS solution. We have experimented with different values of p on several instances from [23]. The best solutions were not generated with one specific value, but for a range of values. Therefore, in each simulation of our algorithm, p is set to a random value between a lower bound ℓ and an upper bound u. This ensures that some simulations have more "exploration power" than others. For our VRP test set, a value between 0.05 and 0.20 appears to give the best results (see Figure 4 for instance `E051-05E`), so we set $\ell = 0.05$ and $u = 0.20$.

Algorithm 1 BINARYMCS-CWS

```
Require: $savingslist$, $r$
  while !$savingslist$.empty() or $solution$.done() do
    $pair \leftarrow savingslist$.pop();
    if feasible($pair$) then
      $processed \leftarrow solution$.process($pair$);
      $yes \leftarrow 0$;   $no \leftarrow 0$;
      for $i = 1$ to $r$ do
        $yes \leftarrow yes +$ SIMULATION($processed$, $savingslist$);
        $no \leftarrow no +$ SIMULATION($solution$, $savingslist$);
      end for
      if $yes \geq no$ then
        $solution \leftarrow processed$;
      end if
    end if
  end while
  if $best$.length() $<$ $solution$.length() then
    return $best$;
  end if
  return $solution$;
```

Algorithm 2 SIMULATION

```
Require: $solution$, $savingslist$
  $p$ = rand($\ell$, $u$);
  while !$savingslist$.empty() do
    $pair \leftarrow savingslist$.pop();
    if rand(0, 1) $> p$ then
      $solution \leftarrow solution$.process($pair$);
    end if
  end while
  if $solution$.length() $<$ $best$.length() then
    $best \leftarrow solution$;
  end if
  return $solution$.length();
```

Note that in both algorithms, ℓ, u and $best$ are global variables. All other variables are local.

Instance	Tightness	Best	CWS	ALGACEA-2	Diff	BinaryMCS-CWS	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%	536	2.10%
`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%
`E101-10C`	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%	1360	3.74%
Total		*12632*	*14866*	*14426*	*14.20%*	*13046*	*3.28%*

Table 2: ALGACEA-2 vs. BinaryMCS-CWS, and their difference ("Diff") with the best known solution.

6 Results

In this section we will compare BinaryMCS-CWS with ALGACEA-2 using a big part of the test sets found at [23]. We used a 3.2GHz machine with 6GB memory and no more than 5 minutes of computation time per instance, which is also the maximum the amount of time spent by ALGACEA-2.

The results are shown in Table 2. The first column lists the instance names, where the first and second number in a name denote the number of customers and the number of vehicles, respectively. The second column denotes the tightness of that instance, followed by the instance's best known solution, found either by an exact algorithm or some algorithm based on metaheuristics. The fourth column shows the obtained solution length by the standard CWS algorithm. Values between brackets denote infeasible solutions, meaning too many vehicles were used. The next four columns represent the obtained solution lengths and their difference with the best known solution, for the ALGACEA-2 and the BinaryMCS-CWS method.

The BinaryMCS-CWS algorithm as described in Section 5.2 was executed with $r = 2000$ and making use of random restarts. It produces solutions with a difference of $13046 - 12632 = 414$ distance units (3.28% on average), whereas ALGACEA-2 has a difference of 1794 (14.20%) from the optimal or best known solution. Interesting to note is that the instances that are hard for the ALGACEA-2 algorithm (such as `E072-04F`) are also relatively hard for our BinaryMCS-CWS method. This is most likely due to to the limitation of the applicability of the CWS method to that particular instance. Apparently, 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 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 expect this ordering to be crucial for obtaining good results. We also point out the simplicity of our method compared to the ALGACEA-2 algorithm. Our experiments suggest that adding more time does not significantly improve our solution quality, which is likely due to the limited applicability of the CWS method. Nevertheless, we think that we have presented an interesting, simple, and well-performing solving method for the VRP. Our obtained solutions are available at [21]. An example solution is given in Figure 2.

7 Conclusion

We have applied Monte Carlo Simulation to the CWS algorithm and developed a solving method for the Vehicle Routing Problem called BinaryMCS-CWS. This method produces solutions with only an 3.28% average deviation from the optimal solutions for a well-known test set, outperforming the previous best known Monte Carlo algorithm for the VRP, and performing comparable to exact and meta-heuristic methods. Even though our method is limited by the applicability of the CWS algorithm on which it is based, using a very simple approach, it produces high-quality results in a reasonable amount of time.

Our algorithm, BinaryMCS-CWS, can most likely still be improved by fine-tuning the value of p, for example by finding a correlation with some domain-specific property of the VRP. Furthermore, it is worth investigating how our method performs within other Monte-Carlo frameworks, such as MCTS [8]. It may also be worth investigating how our method performs on other variants of the VRP, or how Monte Carlo Techniques could perhaps also be applied to other heuristic VRP solving methods.

Acknowledgments
We would like to thank Hendrik Jan Hoogeboom and the anonymous reviewers for their worthy suggestions.

References

[1] B.M. Baker and M.A. Ayechew. A genetic algorithm for the Vehicle Routing Problem. *Computational Operational Research*, 30(5):787–800, 2003.

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

[3] 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*, pages 267–276, 2004.

[4] 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*, 10:63–212, 1986.

[5] G.M. Buxey. The Vehicle Scheduling Problem and Monte Carlo Simulation. *Journal of Operational Research Society*, 30:563–573, 1979.

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

[7] J. Cordeau and G. Laporte. Tabu search heuristics for the Vehicle Routing Problem. In *Operations Research/Computer Science Interfaces*, pages 145–163, 2005.

[8] R. Coulom. Efficient selectivity and backup operators in Monte-Carlo tree search. In *Proceedings of the 5th international conference on computers and games*, pages 72–83, 2006.

[9] G.B. Dantzig and J.H. Ramser. The Truck Dispatching Problem. *Management Science*, 6(1):80–91, 1959.

[10] P.F. de Córdoba, L.M. García-Raffi, A. Mayado, and J.M. Sanchis. A real delivery problem dealt with Monte Carlo Techniques. *Sociedad de Estadistica e Investigacion Operativa Top*, 8(1):57–71, 2000.

[11] J. Dethloff. Vehicle routing and reverse logistics: The Vehicle Routing Problem with simultaneous delivery and pick-up. *OR Spectrum*, 23(1):79–96, 2001.

[12] 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*, pages 2708–2716, 2008.

[13] J. Faulin and A. Juan. ALGACEA-2: An entropy-based heuristics for the Capacitated Vehicle Routing Problem. In *Seventh Metaheuristics International Conference*, 2007.

[14] J. Faulin and A. Juan. The ALGACEA-1 method for the Capacitated Vehicle Routing Problem. *International Transactions in Operational Research*, 15(5):599–621, 2008.

[15] M.L. Fisher. Optimal solution of Vehicle Routing Problems using minimum k-trees. *Operations Research*, 42(4):626–642, 1988.

[16] R. Hassi and S. Rubinstein. On the complexity of the k-customer Vehicle Routing Problem. *Operations Research Letters*, 33(1):71–76, 2005.

[17] G. Laporte. Fifty years of Vehicle Routing. *Transportation Science*, 43(4):408–416, 2009.

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

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

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

[21] F.W. Takes. *Applying Monte Carlo Techniques to the Capacitated Vehicle Routing Problem*, Master Thesis, Leiden University, 2010. `http://www.liacs.nl/~ftakes/vrp/`.

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

[23] D. Vigo. *VRPLIB: A Vehicle Routing Problem LIBrary*. `http://www.or.deis.unibo.it/research_pages/ORinstances/VRPLIB/VRPLIB.html`, accessed May 31, 2010.

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