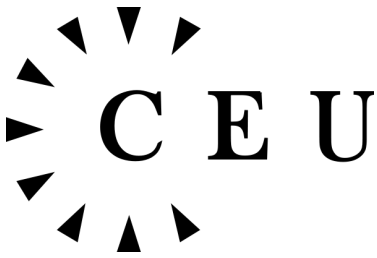
All roads lead to the Depot

Comparison of Methods for Solving the Vehicle Routing Problem

András Dániel Kárpáti

Academic Advisor: István Miklós PhD

A thesis presented for the degree of
Master of Arts in Applied Mathematics



Department of Mathematics and its Applications
Central European University
Hungary

2017.05.09

Acknowledgements

For my parents, who supported me in every possible way in getting to this point.

A very special gratitude to members of the SZMT of Rajk László College for their neverending patience and trust which provided the basis of my life while writing this thesis.

I am also very grateful for Richárd Nagy for using his mysterious ways to help me hold on my sanity in the direst of moments.

Contents

1 Introduction

In my Thesis I will examine the contemporary methods for solving the Vehicle Routing Problem. The Vehicle Routing Problem (VRP) is a nonlinear combinatorial optimization problem. Given a graph and a set of constrains (number of routes, capacity of each route, time windows etc.) one aims to find a least cost set of routes that all start from a given vertex (depot) and that visit every vertex exactly once.

Because of the recent advances in related information technology (geographical software, processing capacity, etc), finding approximate solutions for these problems became a highly researched topic in the field of combinatorial optimization.

Being able to solve this kind of problem in a short time with good performance has many practical applications. These include: planning routes for various logistics services, such as trash companies and pizza delivery services, or designing the optimal method for punching holes on a metal plate.

However, as the VRP is an NP-hard optimization problem, finding exact optimal solutions is practically impossible. Therefore the current focus of the research is to find approximation algorithms that solve the problem in polynomial time with an acceptable performance in terms of cost (time, fuel, robustness etc).

I will describe various heuristic algorithms for solving the Traveling Salesman Problem (TSP) and the Vehicle Routing Problem and test them on different datasets. I will analyze the performance of the algorithms on random examples with different parameters and compare the results.

The rest of the thesis is structured as follows: Section 2 describes the Problem. It contains all relevant definitions and proofs. Section 3 & 4 gives a comprehensive review of the literature in solving the TSP and the VRP. Section 5 discusses various real life case studies of applying VRP solving methods to improve an industrial operation. Section 6 explains my implementation, Section 7 describes experiment design, Section 8 is the computational results and finally Section 9 concludes my findings.

2 Description of the Problem

In this section I will present the mathematical description for the *Travelling Salesman Problem* (TSP) and the VRP.

First of all, all our problems are graph optimization problems, therefore we need some basic graph definitions.

Definition 2.1. *A graph $G(V, E)$ is an ordered triplet made of a set of edges E and a set of vertices V and a set of incidence relations. The incidence relation describes the adjacency of an edge and (1 or 2) vertex(es).(1)*

Definition 2.2. *A graph path is a sequence $x_1, x_2, ..., x_n$ such that $(x_1, x_2), (x_2, x_3), ..., (x_{n-1}, x_n)$ are graph edges of the graph and the x_i are distinct.(2)*

Definition 2.3. *A cycle in a graph G, also called a circuit if the first vertex is not specified, is a subset of the edge set of G that forms a path such that the first node of the path corresponds to the last.(3)*

To emphasize the complexity of the problem, it is important to define the relevant complexity classes. The key point of my thesis is that we are not able to find the exact best solution to the VRP and the TSP because they are NP-complete optimization problems.

Definition 2.4. *A computational problem is in NP if it is solvable in polynomial time by a non-deterministic Turing Machine. (4)*

Definition 2.5. *A problem is NP-hard if an algorithm for solving it can be translated into one for solving any NP-problem problem in polynomial time. NP-hard therefore means "at least as hard as any NP-problem," although it might, in fact, be harder. (5)*

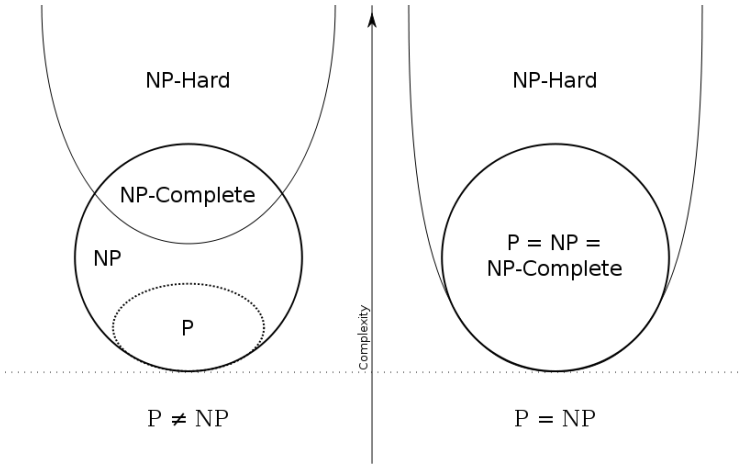
Definition 2.6. *A problem is NP-complete if it is both in NP and NP-hard (any NP-problem can be translated into this problem). (6)*

To date, it is unclear whether P=NP, that is that we can find solutions to NP problems in polynomial time. Figure 1. presents the possible relations of P and NP.

The most basic NP-complete problem is the Satisfiability (SAT) problem.

The SAT is the problem whether a given Boolean formula has a satisfying truth assignment. A Boolean formula has the following parts:

Figure 1: The possible relations of P and NP(7)



- Variables are called literals
- A clause is the disjunction of literals

Definition 2.7. *A Boolean formula is called a Conjuctive Normal Form (CNF) if it is a conjunction of disjunctive clauses.*

Theorem 2.1. *It is possible to construct a Conjuctive Normal Form for every problem in NP and $X \in \mathcal{A}$ problem instance which is satisfiable if and only if the answer for X is yes.*

Proof. I am only going to prove the first part.
Step 1. - SAT $\in$ NP
SAT is in NP because any truth assignment can be verified in polynomial time by a deterministic Turing Machine.

Step 2. - SAT is NP-hard
Now I am going to show that any problem in NP can be converted in polynomial time to a CNF. That means construction a Boolean expression which is satisfiable iff the Turing Machine solving the original problem accepts problem instance. In the following n is the problem size and p is a polynomial function.

To do this we need 3 types of variables:

- $T_{i,j,k}$ is true if cell i on the tape contains symbol j at step k of the computation. We need $\mathcal{O}(p(n)^2)$ of these.

- $H_{i,k}$ is true if the machine's head is at cell i at step k of the tape. We need $\mathcal{O}(p(n)^2)$ of these.
- $Q_{q,k}$ is true if the machine is at state q at step k of the computation. We need $\mathcal{O}(p(n))$ of these.

These variables are organized into a Boolean expression B which is the conjunction of the following:

- $T_{i,j,0}$ for initial contents of the tape. We need $\mathcal{O}(p(n))$ of these.
- $Q_{s,0}$ initial state of the machine
- $H_{0,0}$ initial position of the head
- $\neg T_{i,j,k} \vee \neg T_{i,j',k}$ where $j \neq j'$ meaning that there is at most 1 symbol per tape cell. We need $\mathcal{O}(p(n)^2)$ of these.
- $\vee_{j \in \sum} T_{i,j,k}$ meaning at least 1 symbol per tape cell. We need $\mathcal{O}(p(n)^2)$ of these.
- $T_{i,j,k} \wedge T_{i,j',k+1} \rightarrow H_{i,k}$ where $j \neq j'$ meaning that the tape remains unchanged unless written. We need $\mathcal{O}(p(n)^2)$ of these.
- $\neg Q_{q,k} \vee \neg Q_{q',k}$ where $q \neq q'$ meaning one state at a time. We need $\mathcal{O}(p(n))$ of these.
- $\neg H_{i,k} \vee \neg H_{i',k}$ where $i \neq i'$ meaning one head position at the time. We need $\mathcal{O}(p(n)^3)$ of these.
- $(H_{i,k} \wedge Q_{q,k} \wedge T_{i,\sigma,k} \rightarrow \vee_{(q,\sigma,q',\sigma',d) \in \delta} (H_{i+d,k+1} \wedge Q_{q',k+1} \wedge T_{i,\sigma',k+1})$ which describes possible transitions at step k with the head at position i. We need $\mathcal{O}(p(n)^2)$ of these.
- $\vee_{0 \leq k \leq p(n)} \vee_{f \in F} Q_{f,k}$ meaning that the machine must finish in $p(n)$ steps in an accepting state.

The clauses all have constant sizes (except for the last one, but it also has at most polynomial size). This is important because clauses of constant size can be translated to CNF in constant time. The size of each clause is affected by the problem, while the number of clauses is affected by the size of the problem instance.

If there is an accepting computation for the Turing Machine then by assigning the Q, T, and H variables their intended values then the Boolean expression will be true. On the other hand, if the expression is satisfiable, then there is an accepting computation that follows the steps outline by the truth assignment. There are $\mathcal{O}(p(n)^2)$ Boolean variables and $\mathcal{O}(p(n)^3)$ clauses. Thus the transformation is clearly in polynomial time. □

From this we can also show that the 3-SAT problem is NP-complete.

Theorem 2.2. *3-SAT is NP-complete.*

Proof. Again, the problem is trivially in NP. IF you have a clause longer then 3, then the following transformation is possible from any general CNF clause $(X_1 \vee ...X_n)$:

$$(X_1 \vee X_2 \vee Y_1) \wedge (\neg Y_1 \vee X_3 \vee Y_2) \wedge ... \wedge (\neg Y_{n-3} \vee X_{n-1} \vee X_n)$$

If a clause has length n, then the number of new clauses is $n-3$. This new clause is satisfiable iff the CNF is satisfiable. If the original CNF is satisfiable, then one of the X_i literals must be true, and therefore the 3-SAT is true. On the other hand, if the 3-SAT is true, then one of the X_i literals must be true. Suppose otherwise, that all X_i are false and the 3-SAT is true. This means that Y_1 must be true. This means that Y_2 must be true and so on. This means that Y_{n-3} is true. But in this case the 3-SAT is not satisfied, therefore one of the X_i listerals must be true. □

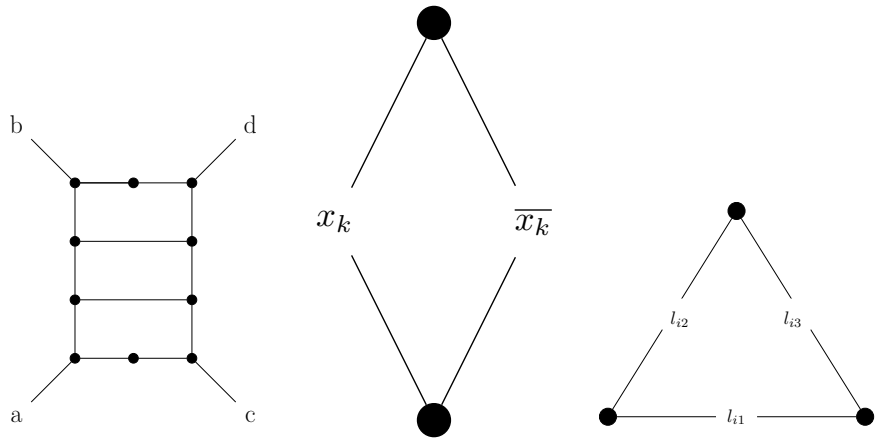
As we will see, the VRP and the TSP are both computational hard problems, since they are at least as hard as finding a Hamiltonian circuit.

3 Travelling Salesman Problem

The aim of the Thesis is to present solutions for the VRP. However, to achieve a comprehensive understanding of the topic we must begin with the TSP, as the VRP is the generalization of the TSP.

In a TSP, the problem is as follows: given a graph G and a distance matrix C_{ij} (usually interpreted as physical distance or time) one aims to find a least cost Hamiltonian cycle.

Definition 3.1. *Hamiltonian cycle A Hamiltonian cycle is a cycle in a graph that visits every vertex exactly once.(8)*



(a) XOR gadget (b) Variable gadget (c) Clause gadget

Figure 2: Gadgets used in the proof

Theorem 3.1. *The Hamiltonian circuit problem is NP-complete*

Proof. **Step 1.** The Hamiltonian circuit problem is in NP. To prove that the problem is in NP, we need to show that if there is a solution, we can check it in polynomial time. In this case, this is pretty straightforward, as we get a list of vertices, and we need to check whether there are edges between the given vertices. This can be done in $O(n^2)$ time, therefore the problem is in NP.

Step 2. The Hamiltonian cycle is NP-complete.

To see this, I am going to show that if you have a black box (also called an oracle) that solves the Hamiltonian cycle problem in polynomial time, then you can solve the 3-SAT problem as well. By the above proven theorem we know that the 3-SAT is NP-complete

First, we wish to create a graph G such that a 3-SAT instance F is satisfiable iff G has a Hamiltonian cycle.

The idea of this step is that we construct "gadgets" that represent logical operators.

Figure 2 is a graphical representation of an xor gate. Note that a Hamiltonian cycle must either enter and leave at a and b or c and d by the next Lemma.

I am going to refer to the big black nodes from now on as supernodes to distinguish them from the nodes in the exclusive or gadget.

The graph construction procedure is as follows:

1. For each variable construct a variable gadget.
2. For each clause construct a clause gadget
3. Connect every supernode to every supernode
4. If a clause c_i contains a literal $l_i = x_i$ then attach a and b of an exclusive or gadget to a variable gadget and attach c and d to the clause gadget. If a variable occurs in more than one clause, then attach two xor gadgets in series where b of one gadget is attached to a of the next.

The procedure is clearly in polynomial time.

Lemma 3.2. *Because of the construction of the graph the XOR gadget can be visited through a and b nodes or c and d nodes but not in any other combination in a Hamiltonian Circuit.*

Proof. The path can enter through b or d. I am going to show that if the path enters through b, it must leave through a and not c or d. The proof is symmetric for the other side.

If the path enters at b it is impossible to leave through d since the vertex on the right of b will be left out.

Therefore the path must start to the right to d. Then one vertex down. The path cannot go down, since it would not be able to visit the node to the left. Therefore it must go to the left.

Repeating the above argument yields us the result that the path must leave through a. □

Suppose there is a valid truth assignment, that is there is a true literal in every clause. Starting from the first variable node the Hamiltonian path passes through the a and b nodes of each XOR gadget for which the corresponding variable is true. If the variable is false, just connect the two variable supernodes with skipping the XOR gadget.

After the last variable supernode, the circuit visits the first clause supernode. If the variable corresponding to the literal is true, then go to the next supernode, otherwise go through the XOR gate.

Not that it is possible to visit every node of a clause and the left-out XOR gadgets and then go on to the next clause *iff* there is at least one true literal in each clause.

Therefore if there is a valid truth assignment, then there must be a a Hamiltonian Cycle.

On the other hand, if there is a Hamiltonian Cycle, then all vertices are visited, therefore all XOR gates are visited. If all XOR gates are visited, then there must be a true literal in each clause. If every literal is false in a clause, then all 3 XOR gadgets must be visited that are connected to the clause through the clause supernodes which is a cycle in itself.

□

Now we can see why the Travelling Salesman problem is also hard to solve. It is NP-hard as an optimization problem. However, if stated as a decision problem such that "is there a HC in this graph at most x length?" it is NP-complete. I am going to prove the latter.

Theorem 3.3. *The Travelling Salesman Problem as a decision problem is NP-complete.*

Proof. **Step 1.**
It is evident that given a solution we can check in n steps if it is indeed a HC in the graph and summing the cost and comparing it to the bound is not dependent on problem size. So the TSP in in NP. **Step 2.**
I need to show that if I can solve a TSP in polynomial time, then I can solve an other NP-hard problem in polynomial time. If I can solve a TSP in polynomial time, then I can also solve the HC problem, therefore TSP is NP-hard.

□

We can see in the same way that the Vehicle Routing Problem is NP-complete. Since if we can solve a VRP in polynomial time, we just need to limit the number of routes to 1 and declare one of the vertices as the Depo.

3.1 Definition of the TSP

There are many ways to define an instance of a TSP. I am going to present 2 integer programming formulations and describe their relative strengths and weaknesses. This chapter is mostly based on Pataki (2003)(9).

3.1.1 DFJ formulation

This is one of the earliest formulations and it is due to Dantzig, Fulkerson and Johnson (1954)(10). It defines x_{ij} for every nodepair of nodes and $x_{ij} = 1$ iff

the ith and jth nodes are subsequent in the optimal path. The distance matrix C_{ij} is given. Target: minimize the following expression:

$$\sum_{i \neq j} c_{ij} x_{ij} \tag{1}$$

Subject to the following constrains:

$$\sum_{j=1}^{n} x_{ij} = 1 \tag{2}$$

$$\sum_{i=1}^{n} x_{ij} = 1 \tag{3}$$

$$\sum_{i,j \in S} x_{ij} \leq |S| - 1$$
$$S \subset V, 2 \leq |S| \leq n - 2 \tag{4}$$

Constraint 2 and 3 specify that each node most have 1 as in and outdegree as well. Constraint 4 is called a subtour elimination constraint.

If 4 was not present, we would be looking for a cycle cover of the graph, so many unconnected cycles would be allowed. So in any subgraph there must be one less edge than vertex.

3.1.2 MTZ Formulation

We can replace 4 by adding variables u_i such that

$$\begin{aligned} u_1 &= 1, & \\ 2 \leq u_i &\leq n & \forall i \neq 1, \\ u_i - u_j + 1 &\leq (n-1)(1 - x_{ij}) & \forall i \neq 1, \forall j \neq 1. \end{aligned} \tag{5}$$

The above equations force $u_j \geq u_i + 1$ if $x_{ij} = 1$. If there are more than one tour on the vertices, then one cycle does not contain node 1, and in that cycle u_i values have to increase to infinity.

The strength of the MTZ formulation compared to the DFJ formulation is that the DFJ formulation contains exponentially many subtour elimination constaints (1 for every subset of V), the MTZ formulation only contains a poly-

nomial number of equations.

However, most research is still based on the DFJ formulation. One mitigated the problem of exponentially many constraints by recognizing the fact that not all subtour elimination constraints need to be put into formulation from the start. They can be generated as needed.

3.2 Solving the TSP

After describing the TSP in a mathematical way, the next chapter is going to describe the algorithms for solving the problem. As we have seen the TSP is an NP-hard combinatorial optimization problem, therefore it is impossible to solve it to optimality in large problem instances. Therefore I am going to present just one exact algorithm and more heuristics.

Heuristics can present a feasible (though not the optimal) solution for the TSP in polynomial running time. The heuristic approach is the only way to deal with the TSP in real life problems.

3.2.1 Branch and Bound Algorithm

BB is an exact method. The point of the BB algorithm is to relax the constraints of the problem, and give a structure to the search space. In our case, this is a tree structure. The algorithm performs a search in the space of all optimal solutions along branches of the tree. The algorithm also computes a lower and an upper bound on each branch, and a branch is discarded if it cannot produce a better solution than the current best.

3.2.1.1 CMT algorithm

Now I am going to describe the algorithm of Carpaneto, Martello and Toth (11). The CMT is a Branch and Bound method. In this case the subtour elimination constraints are relaxed and get an Assignment Problem. It is an Assignment Problem in the sense that every vertex must be followed by a vertex and preceded by a vertex. If we relax the subtour elimination constraints, we - surprisingly - allow the construction of subtours, and we get a vertex cycle cover.

Definition 3.2. *A vertex cycle cover is a set of cycles in a graph whose subgraphs contain all vertices.*

The CMT Algorithm uses the following notation:

- z^* the cost of the best solution so far
- z_h value of the target function at point h of the search space
- $\underline{z_h}$ is the lower bound on z_h
- I_h the set of edges in the current solution
- E_h the set of edges not in the current solution

The steps of the algorithm are these:

1. Find a feasible solution z^* for the TSP.
2. Solve the AP with $I_h = E_h = \emptyset$. If $z_1 \geq z^*$ then stop. z_1 is the root of the search space.
3. Stop if there are no more vertices in the search space. Otherwise pick the next solution
4. Since the solution in h is not feasible, we need to solve z_r subproblems where z_r is the number of branches. These are characterized by the sets I_{h_r} and E_{h_r}. Pick the cycle that contains the least amount of edges not in I_h. These are denoted $(i_1, j_1)...(i_s, j_s)$. Thus we get the following subproblem:

$$I_{h_r} = \begin{cases} \mathrm{I}_h, & r = 1 \\ \mathrm{I}_h(i_u, j_u) & u = 1...r-1 \quad r = 2...s \end{cases}$$

The lower bounds on the branches are calculated from the AP solution. If the solution of the AP has higher cost then the current best TSP solution, then there is no reason to further explore that branch.

In the worst case, the running time of the algorithm is the same as the number every possible I_h set of $n-1$ edges. In a complete graph of size n there are $\mathcal{O}\binom{n}{2}$ edges, and therefore there are $\binom{\binom{n}{2}}{n-1}$ possibilities of that, which is an exponential function of n.

CMT is an early approach to solve the TSP to optimality. The current record is an instance of 85,900 cities (12). It is an open problem whether it is possible to solve the TSP with complexity $\mathcal{O}(c^n)$ with $c < 2$ (13).

3.3 Spanning Tree Algorithm

This is a heuristic algorithm which means that it runs in polynomial time, but will not produce an optimal solution.

The idea is to find a Minimum Spanning Tree, then construct a tour using it.

It is possible to find a MST using Kruskal's Algorithm in $O(n\ln(n))$ where n is the number of edges in the graph. Kruskal's algorithm is the following(14): For a graph $G(V,E)$ construct $G'(V,\emptyset)$. Arrange E in increasing order of weights. Then repeat the following:

1. If G' is a spanning tree, stop
2. Pick the next edge from E
3. If adding it to G' does not create a cycle then adding it to G'. Otherwise go to Step 1.

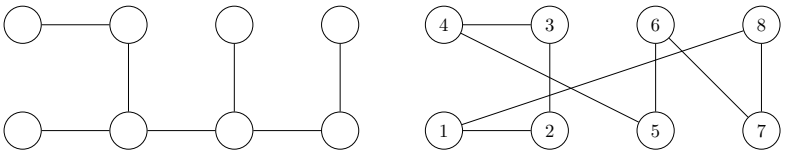
Theorem 3.4. *On a connected weighted graph G Kruskal's Algorithm results G', a minimum weight spanning tree.*

Proof. **Step 1.**: G' is a spanning tree

- G' is a forest, no cycles are created
- G' is spanning: Suppose there is a vertex v which is not incident to any edges of G'. Since G is connected, an edge which is incident to v must have been considered. It could not have created a cycle, therefore the existence of v contradicts the definition of G'.
- G is connected: Suppose otherwise. Then there is a path in G between any 2 components of G' since G is connected. Then one of these edges must have been considered for adding to G'. It could not have created a cycle, therefore the existence of more than 1 component contradicts the definition of G'.

Step 1.: G' is a minimal
I am going to prove this by induction. Let $G*$ be a minimum spanning tree. Suppose $G' \neq G*$. Then $\exists e \in G* \setminus G'$ edge of minimum weight. And $G' \cup e$ contains a cycle C such that:

1. Every edge in C has weight at most $w(e)$
2. $\exists f \in C \setminus G*$



(a) MST (b) Tour

Figure 3: The MST and the corresponding tour according to the algorithm

Consider $G'' = G' \setminus e \cup f$. The total weight of G'' is at least the total weight of G' and it has more edges in common with $G*$. We can repeat this process until we get $G^n = G*$. But $G*$ is minimum, therefore $w(G') = w(G*)$, so G' is minimum as well. □

Then the algorithm for the TSP is as follows:

1. Pick a leaf i_0 on the MST, $i := i_0$.
2. Pick a neighboring edge (i, j) and add it to the tour. Let $i := j$ and repeat until there are unvisited neighboring edge.
3. If Step 2 cannot be repeated, go back to k, the last node before i, $i := k$ and do Step 2 again.
4. If $i = i_0$ the algorithm stops.

3.4 GENIUS Algorithm

This algorithm was developed by Genreau, Hertz and Laporte (1994)(15). It is made up of an insertion procedure (GENI) and a postoptimization procedure (US).

3.4.1 GENI

GENI stands for Generalized Insertion Procedure. The idea is that the algorithm can insert a vertex to a route between vertices that are not necessarily adjacent in the route.

Suppose we want to insert vertex v between vertices v_i and v_j, v_k is a vertice between v_j and v_i, while v_l is between v_i and v_j considering the orientation of the tour. On Figure 4 v_i is 1, v_j is 5, v_k is 3.

Type I. insertion

Here $v_k \neq v_i$ and $v_k \neq v_j$.

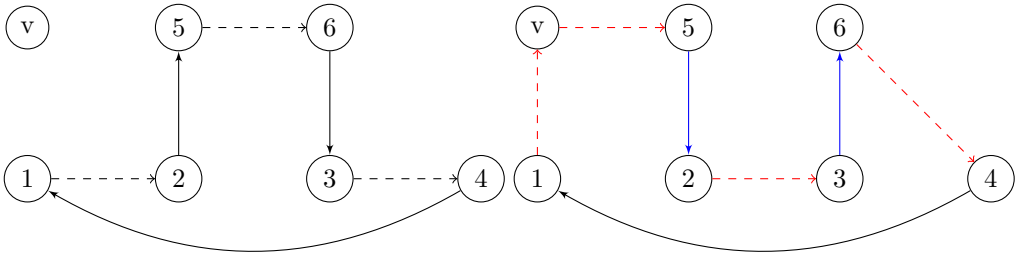
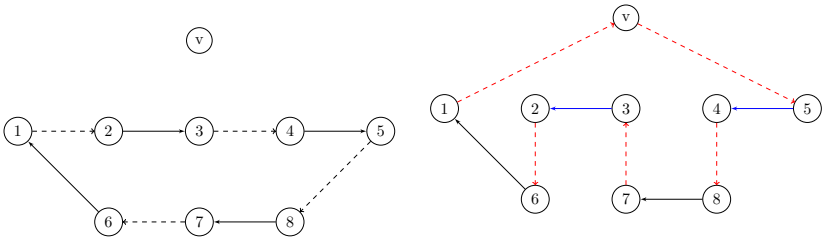


Figure 4: Type I. Insertion



(a) Candidate for Type II Insertion (b) Type II Insertion

Figure 5: Type II. Insertion

- Arcs deleted: $(v_i, v_{i+1}), (v_j, v_{j+1}), (v_k, v_{k+1})$
- New arcs: $(v_i, v), (v, v_j), (v_{i+1}, v_k)$
- Paths reversed: $(v_{j+1}...v_j), (v_{j+1}...v_k)$

The insertion procedure is on Fig. 4. HEre v_i is 1, v_j is 5, v_k is 6, v_j is4.

We wanted to insert v between vertices 1 and 5. Dashed lines represent single edges, while full lines represent paths made up of (possibly) multiple edges. Red edges are new edges, blue paths are reversed compared to the original.

Type II. Insertion Here the condition is that $v_k \neq v_j$, $v_k \neq v_{j+1}$, $v_l \neq v_i$, $v_l \neq v_{i+1}$

- Deleted arcs: (v_i, v_{i+1}), (v_{l-1}, v_l), (v_j, v_{j+1}), (v_{k-1}, v_k)
- New arcs: (v_i, v), (v, v_j), (v_{l-1}, v_l), (v_l, v_{j+1}), (v_{k-1}, v_{l-1}), (v_{i+1}, v_k)
- Paths reversed: $(v_{j+1}...v_j), (v_{j+1}...v_k)$

An example is on Figure 5.

3.4.2 US procedure

The US procedure takes a solution to the TSP and tries to improve it. It has 2 parts: Unstringing and Stringing. Unstringing means removing a vertex from the path, Stringing means reinserting it. The US procedure can use any TSP solution, not only the ones produced by GENI. Unstringing uses the reverse of moves described as Type I. and Type II. removals, while strining uses them as insertion procedures.

4 Vehicle Routing Problem

4.1 Formalizing the VRP

I am going to present a formal description of the VRP as an integer programming problem, just like the TSP. Given a graph $G(V, E)$ with a *Depo* at vertex 1 and an adjacency matrix $C(i, j)$, our goal is to find a least-cost set of routes satisfying the following constraints:

- Every vertex is visited by exactly 1 route
- Every tour starts and ends at the Depo.

In a typical VRP there can be many more constraints, such as:

- Capacity of a route
- Time windows at each vertex
- Route length / time constraints.
- Maximum number of routes

However, in this thesis I am only going to analyze the case where there is a capacity and a length constraint.

5 Algorithms for solving the VRP

5.1 Naive algorithm

This is a theoretical concept for showing the number of maximum steps for solving the VRP. The naive algorithm will cycle through all possible solutions, and pick the best one in the end.

Given n vertices, we have at most $\binom{n}{2}$ edges. We must group these edges into as many groups as there are vehicles starting from the depot, then order each group. This results in approximately $2^{\mathcal{O}(n^2 \log n)}$ possibilities.

5.2 Exact Algorithms

For completeness of this paper, I am going to briefly describe the most common techniques for solving small VRP instances. These methods have little applied significance, as the most common VRP's are far to big for solving for optimality.

These methods can be categorized as follows:

- **Direct Tree Search methods**
  These are mostly built on the Branch Bound technique. The idea is to exclude as many unfeasible solutions as possible

- **Dynamic Programming**
  Dynamic Programming solves a complex problem by dissecting it to smaller and easier problems. These subproblems occur repeatedly in solving the original problem, therefore we can save time by solving them just once.

- **Integer Linear Programming**
  ILP means that given a linear function to minimize and a set of constraints the goal is to satisfy all constraints with only integer values.

5.3 Applied Algorithms for solving the VRP

These algorithms will not solve the TSP for optimality, but use a heuristic for getting an approximate solution. The advantage of these methods is that they run in polynomial time. Different heuristics perform differently on different datasets, and the aim of this thesis is to analyze these differences.

5.3.1 Clarke & Wright Algorithm

This is one of the first known heuristics for solving the VRP by Clarke and Wright (1964)(16). In the CW algorithm we consider the number of routes as an optional constraint.

Starting Solution

Suppose we have as many vehicles as vertices. Every tour is made up for going from the Depo to a vertex and back.

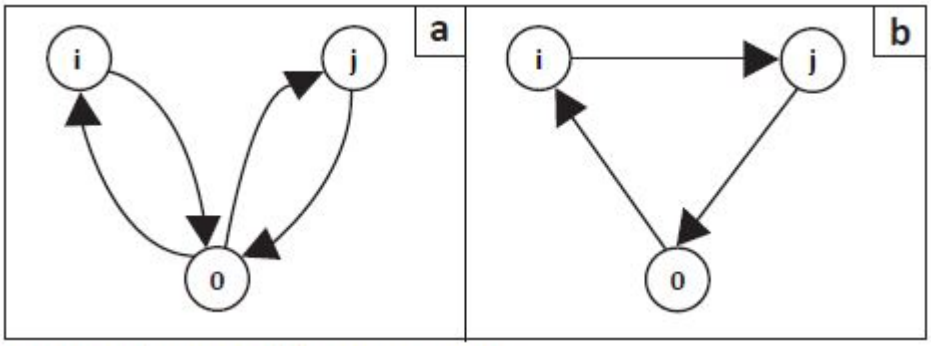


Figure 6: CW Savings Algorithm (17)

Saving

The algorithm goes on with improving the original solution by concatenating routes. The algorithm calculates for every pair of routes the amount of time / distance saved by merging them. Thus, we can create a list of moves. The algorithm will keep executing these moves until is it not possible anymore to execute a move without violating a constraint. The algorithm runs in $\mathcal{O}(n^2 \ln n)$ time.

5.3.2 Sweeping Algorithm

This algorithm is by Wren Alan (1972)(18). Here the number of vehicles must be specified as a constraint. The steps of the algorithm are as follows:

Clustering and Starting solution

The algorithm uses polar coordinates instead of an adjacency matrix for clustering the vertices. The Depo is $(0, 0)$, we arbitrarily choose a starting angle ϕ. We make a list of the vertices by *sweeping* the plane with a halfline starting from the Depo with angle ϕ.

Once we have this ordered list, the algorithm adds the first vertice to the first cluster. The algorithm adds the second vertice to the first cluster if the tour defined is better then just adding it to a new cluster. When adding a vertice to a cluster the algorithm checks all relevant constraints and will not violate them.

It is possible that 2 vertices that are quite distant to get to the same cluster. We can avoid this by setting a constraint on the maximum length of a step. If this is violated, the algorithm will start a new cluster.

Improvement Heuristics

There are 5 such heuristics:

- **Inspect**
  Removes cycles from each route

- **Single**
  Checks if any vertex can be moved to an other route for cost reduction. The algorithm uses the first possibility, it stops searching if it finds an opportunity

- **Pair**
  Pair removes a vertex from a route, if an other vertex can be inserted by single.

- **Complain**
  This is useful if some vertices are left out from the solution. The heuristic checks all routes and vertices for an opportunity to remove one and insert the inspected vertice. Then it tries to find a new spot for the removed vertice.

- **Delete**
  After the abovementioned procedures the algorithm tries to delete a route by adding its vertices to other routes.

5.3.3 TABUROUTE

This algorithm is the work of Gendreau, Hertz, Laporte (1994) (19). TABUROUTE uses a *metaheuristic* called TABU search. A metaheuristic is a heuristic that can be used to solve different computational problems, not just one specific problem. The idea behind TABU search is that when vertices are moved between routes several solutions are revisited and the exploration of the state space often revisits certain solutions. Therefore some moves are declared tabu thereby making the search procedure explore more of the search space. So for a Tabu Search strategy we need the following:

- A *forbidding strategy* which manages what goes into the tabu list

- *a freeing strategy* which removes solutions from the tabu list

- *a short-term strategy* which describes situations when the algorithm ignores the tabu list and a selection strategy which chooses the next possible solutions

- *stopping criterion*

Several steps of the algorithm are based on the GENIUS heuristics (as a selection strategy) by the same authors. The algorithm is designed to solve the following version of the VRP:

- Given a directed graph $G(V, A)$ where V denotes the set of vertices and A denotes the set of arcs, v_0 is the depot and there is a cost c_{ij} associated to each (v_i, v_j) arc..
- every route starts and ends in the depot and has length at most L
- every vertex is visited once
- there is a demand q_i associated with every vertex and the capacity of each route is Q
- the final solution must satisfy the length and capacity constraints for each individual route

Additional notation:

- R_i a route, m is the number of routes
- the objective function is $F_1(S) = \sum_r \sum_{R_r} c_{ij}$ for a feasible solution, and

$$F_2(S) = F_1(S) + \alpha \sum_r \left[\left(\sum_{v_i \in R_r} q_i \right) - Q \right]^+ \beta \sum_r \left[\left(\sum_{(v_i, v_j) \in R_r} c_{ij} + \sum_{v_i \in R_r} \delta_i \right) - L \right]^+$$

  is the objective function for unfeasible solutions which incorporates penalty terms for violation route length and capacity constraints. The + signs mean positive part of a number.

- f_v number of times vertex v has been moved
- δ_{max} is the largest observed difference between values of $F_2(S)$ at two iterations

The algorithm uses the procedure SEARCH(P) where P is a vector of parameters including

- W vertices allowed to be moved from their routes

- q number of vertices candidate for reinsertion
- p_1, p_2 neighbourhood sizes for search
- θ describes the number of iterations for which a move is TABU
- h frequency of α and β terms
- n_{max} the number of steps until the procedure runs without improving the objective function

The SEARCH procedure is as follows:

1. Randomly select q vertices from W
2. *Evaluation of moves* **Repeat this for all selected vertices**
   Consider all moves for v to routes containing at least one of its p_1 neighbours
   (a) Calculate the cost of inserting v to R_s using GENI with p_2 neighbourhood size and get solution S'.
   (b) If the move is tabu, disregard it unless it improves the current best feasible solution
   (c) Otherwise assign $F(S')$ to $F_2(S')$ if $F_2(S') < F_2(S)$ or to $F_2(S') + \delta_{max}\sqrt{m}gf_v$ otherwise
3. Select the best move
4. Apply the US procedure to the solution if certain conditions are met
5. If US has not been used then undoing the move is considered tabu for θ steps
6. Update penalty terms α and β. The aim is to get a mix of feasible and unfeasible solutions
7. Termination check

This describes the SEARCH procedure. The TABUROUTE algorithm consists of constructing an initial solution and calling SEARCH multiple times with different parameters.

5.3.4 MCMC Methods

Metropolis - Hastings Algorithm

The Metropolis algorithm is a sampling method, not an algorithm for solving the VRP. However, this algorithm is a useful tool for sampling arbitrary distributions (for example to get a sample from all possible MST's of a graph), which is crucial for my next metaheuristic, Simulated Anealling. I describe the algorithm based on Chib et al. (2008)(20), but the algorithm was originally developed by Metropolis et al.(1953) (21) and later generalized by Hastings (1970)(22).

The Metropolis-Hastings algorithm is a Markov Chain Monte Carlo method in which we have a target distribution $\pi(x)$ from which we want a sample. Usually, in an MCMC method we have a transition kernel $P(x, A)$ which represents the probability of being in x and moving to the set A, and we are interested in finding the stationary distribution. In this case, we are given a target distribution and the goal is to design the kernel, to desing a chain which converges to the target distribution. The invariant distribution satisfies

$$\pi^*(dy) = \int P(x, dy)\pi(x)dy$$

where $\pi^*(dy) = \pi(y)dy$ and the nth iteration of the transition kernel is $P^{(n)}(x, A) = \int P^{(n-1)}(x, dy)P(y, A)$.

Suppose the transition kernel has the form for some $p(x, y)$:

$$P(x, dy) = p(x, y)dy + r(x)\delta_x(dy)$$

where $r(x)$ is the probability of remaining in x and $\delta_x(dy) = 1$ if $x \in dy$ and 0 otherwise. If $p(x, y)$ satisfied the reversibility condition

$$\pi(x)p(x, y) = \pi(y)p(y, x)$$

then $\pi(x)$ is invariant under $P(x, A)$.

To find $p(x, y)$ such as this, suppose we have a candidate generating density $q(x, y)$ which can be simulated by some known method, where $\int q(x, y)dy = 1$, x is the current position of the chain and y is the proposed step. We usually find from some x that

$$\pi(x)q(x, y) = \pi(y)q(y, x)$$

To fix this, we introduce $\alpha(x, y) < 1$ and the probability of a move will be

$$p_{MH} = q(x, y)\alpha(x, y)$$

. We want to satisfy

$$\pi(x)q(x, y)\alpha(x, y) = \pi(y)q(y, x)$$

Therefore the definition of α is the following:

$$\alpha(x, y) = \begin{cases} \min\left[\frac{\pi(y)q(y,x)}{\pi(x)q(x,y)}, 1\right] & \text{if } \pi(x)q(x, y) > 0 \\ 0 & \text{otherwise} \end{cases}$$

To complete the kernel, $r(x)$ is the probability of remaining at position x and is defined as

$$r(x) = 1 - \int q(x, y)\alpha(x, y)dy$$

The resulting algorithm for and N element sample from a distribution $\pi(x)$ is this

1. Repeat for $j = 1...N$
2. Generate y from $q(x^{(j)}, \cdot)$ and u from $\mathcal{U}(0, 1)$
3. If $u \leq \alpha(x^{(j)}, y)$ then $x^{(j+1)} = y$ and $x^{(j+1)} = x^{(j)}$ otherwise
4. Return $x^{(1)}, ...x^{(N)}$

The M-H algorithm has a large number of practical applications, including for example simulating Bayesian posterior distributions, but now I am only going to focus on Simulated Anealling.

5.3.5 Simulated Annealing

SA is a randomized local search procedure first proposed by Kirkpatrick (1983) (23) as an analogy for annealing process of solids. In physics, annealing is the process for obtaining low-energy states of solids, by first heating them up, then cooling them. When the temperature is high, the solid melts, allowing the particles to move freely, and during the cooling the particles arrange themselves in a low-energy state. SA is implementing this technique in optimization.

Suppose you want to maximize a target function $h(\mathbf{x})$. This is the same as minimizing $\exp(-h(\mathbf{x})/T)$. As T tends to zero, the probability distribution $\pi_T \approx \exp(-h(\mathbf{x})/T)$ puts more of its mass to the vicinity of the maximum of h.

While most local search methods only accept a next solution if it is better than the current best, SA accepts a worse solution with a given probability. This is crucial, since local search methods are often stuck in local optima. The above-mentioned probability is determined by a control parameter called temperature T which tends to zero according to a cooling function.

Now I am going to follow the description of Osman (1993)(24). A specification requires the following:

- Starting and final temperatures
- rule for updating temperatures after each iteration
- update rule for temperature reset after the system "freezes"
- Stopping criterion
- a way to generate neigbouring solutions

According to Osman(24) the best calibration includes a non-monotonic cooling schedule and a systematic (as opposed to random) neighbourhood search. For example it is worth it to limit inserting a vertex to a new path to paths which are already close to the vertex.

The algorithm is as follows:

1. Generate an initial solution by a heuristic
2. Perform a test cycle to determine the starting and freezing temperatures and an estimate of the number of feasible exchanges
3. Select a solution S' and compute $\delta = C(S') - C(S)$
4. If $\delta < 0$ or $u \leq \exp \frac{-\delta}{T_k}$ where u is uniform random in $[0, 1]$ then accept the new solution. Check if it is a new best solution, and register the temperature if it is (T_b)
5. Update temperature according to the cooling schedule
6. Stop if stopping criteria is met, otherwise go to step 3

5.3.6 Parallel Tempering

Parallel Tempering is a generalized case of Simulated Annealing. Roughly speaking, PT runs a number of SA's on constant temperature in a parallel method, and the individual SA chains are allowed to switch states.

PT was first proposed by Geyer (1991)(25) in a conference proceedings. Instead of $\mathcal{X} \times \mathcal{I}$ as in SA ($\mathcal{X}$ is the search space, $\mathcal{I}$ is the iterations), PT uses $\mathcal{X}_1 \times ... \times \mathcal{X}_I$ where all $\mathcal{X}$ are identical copies of each other. We define a joint probability distribution on the product space as

$$\pi_{pt}(x_1...x_I) = \prod_{i \in I} \pi_i(x_i)$$

and run parallel MCMC schemes on each of the $\mathcal{X}_\rangle$. The algorithm is defined as follows:

- The current state is $(x_1^{(t)}...x_I^{(t)})$, draw u Uniform$[0, 1]$.
- If $u \leq \alpha_0$ we conduct an M-H step and update every state.
- If not, the algorithm proposes a swapping step: it picks two states at random, and proposes that they are switched. If the higher temperature state has less energy, then the swap is automatically accepted. Otherwise, draw q Uniform$[0, 1]$. If

  $$q < \frac{\pi_i(x_{i+1}^{(t)})\pi_{i+1}(x_i^{(t)})}{\pi_i(x_i^{(t)})\pi_{i+1}(x_{i+1}^{(t)})}$$

  then accept the swap, otherwise keep current state and go back to step 1.

6 Implementing the algorithms

In this Section, I focus on the practical implementation of the algorithms discussed in the previous Section. I will focus on the more complicated algorithms only.

In my experiment, I solve VRP's with length and capacity constraints. The more complicated algorithms, especially the MCMC algorithms are likely to mostly produce unfeasible solutions and eventually get lost in the search space. It is important to note that limiting the search space to only feasible solutions can seriously limit the algorithms ability to explore the search space, thererfore

it is not an option. So I needed a scheme to find feasible solutions in the neighborhood of unfeasible solutions.

For this, I used the US postoptimization algorithm. If the procedures failed to produce feasible solutions in terms of the length constraint in n (arbitrarily selected) iterations, I used US to shorten the individual paths. If the result was feasible, then the algorithm could proceed. If not, switch the state to the best feasible solution and proceed. With this scheme the algorithms did not get stuck in local optima with a good choice of n. In my experience n is typically between 1/5 and 1/10 of maximum iterations.

6.1 Tabu Search

Tabu Search requires lots of parameters. The cited article gives guidelines for tuning these parameters, however these guidelines were quite unfeasible for me. On one hand, the quality of the solutions were not particularly good, and the computational resources required for solving VRP with 90 vertices were impossibly high. I used only 1 call for SEARCH and limited the search neighborhood compared to the article. This gave me an acceptable performance and running time.

The typical behaviour of Tabu Search is on the Figure 7.

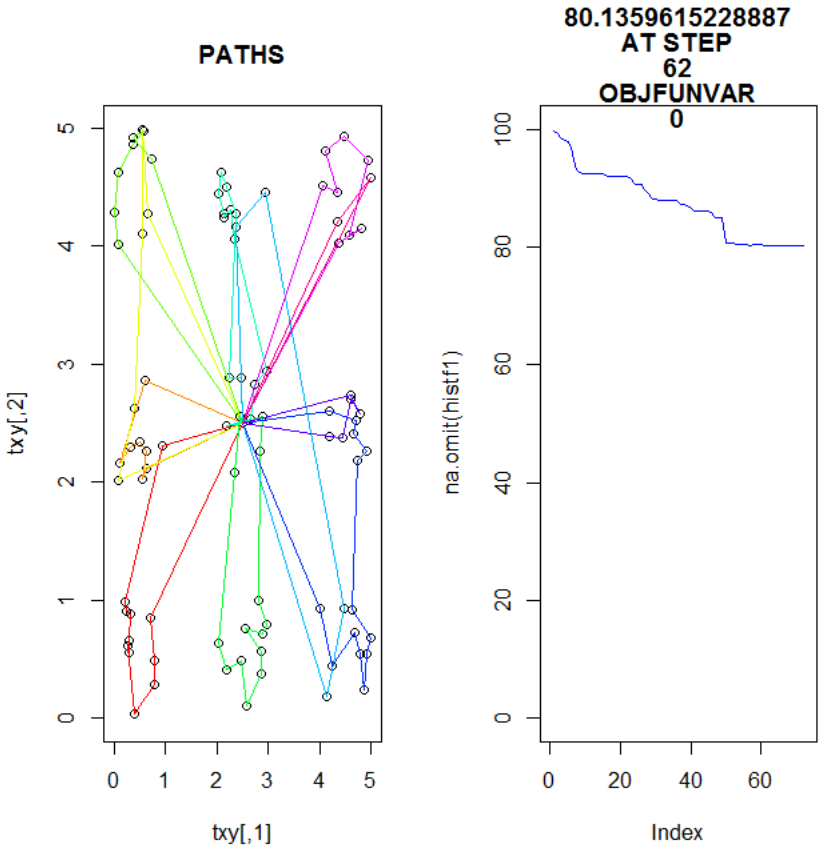
We can see that most of the improvement takes place in the first 2/3 of the running time. I also needed to modify the stopping criterion compared to the article. Using that prduced long running times with no improvements. I used a stopping criteria, where if the variance of the objective function decreased below 2%, the algorithm would stop. The key here is to find the optimal resource/quality ratio.

6.2 Simulated Annealing

SA requires relatively little parametrization, the most important settings are the cooling schedule and a returning criterion. I used a sinusoid cooling schedule. Interestingly, the value of the objective function followed the value of the temperature. However, this is no surprise as at higher temperatures the objective function is allowed to increase more, while on lower temperature the objective function has a strong negative drift. I used 5 waves in the temperature function for optimal results.

SA was prone to getting stuck and not producing feasible solutions. If the algorithm could not produce a feasible solution after 500 successive iterations,

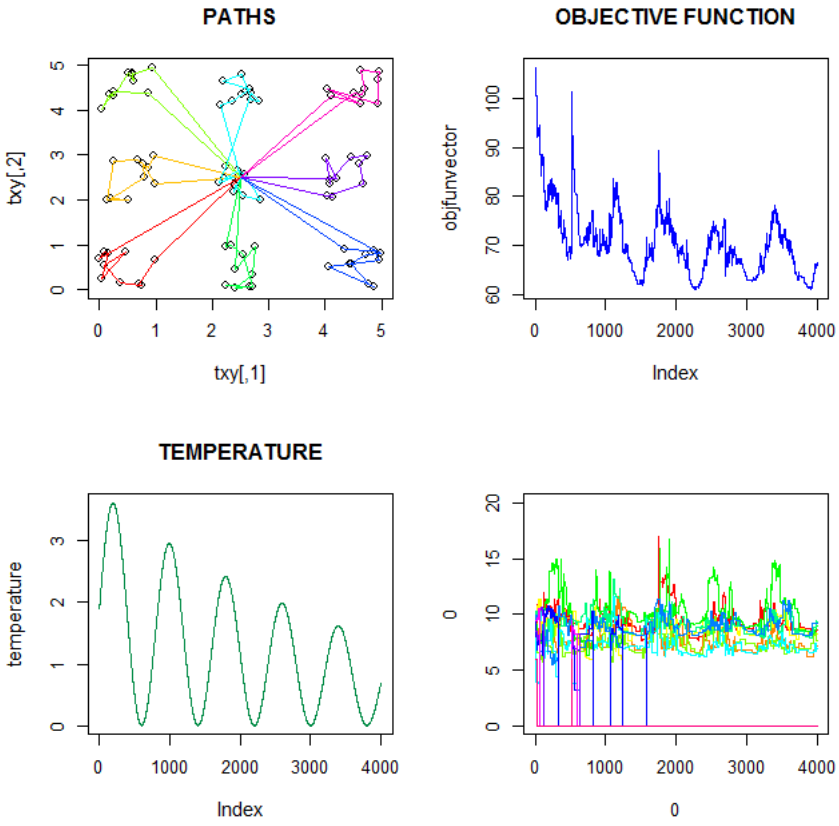
Figure 7: Typical behaviour of Tabu Search. Note that most of the improvement takes place in the early steps, the benefit of each successive iteration decreases while its cost remains the same. Image produced by R.



it tried to make the current solution feasible with US. If the attempt was unsuccessful, the state is switched to the best feasible state. For being stuck the number of successive iterations was 20.

As a proposal distribution I used the heuristics of GENIUS and a simple remove and a simple insertion. Using the Type 1 and Type 2 insertions of GENIUS significantly improved the performance of both SA and PT. When inserting a vertex, I used the p-neighborhood of a vertex (the same size for SA, PT and Tabu Search), instead of just random search. I allowed the SA proposal procedure to investigate 10 random insertions and removals for each vertex. Since Tabu Search explores all p^4 options, and p is usually around 5,

Figure 8: Typical behaviour of Simulated Annealing. Note that the value of the objective function follows the value of the temperature closely. You can also see where the algorithm jumped back to a previous feasible state. Also note that most of the time SA is not producing feasible solutions because of individual pathlengths. Image produced by R.



this is still quite random compared to Tabu Search.

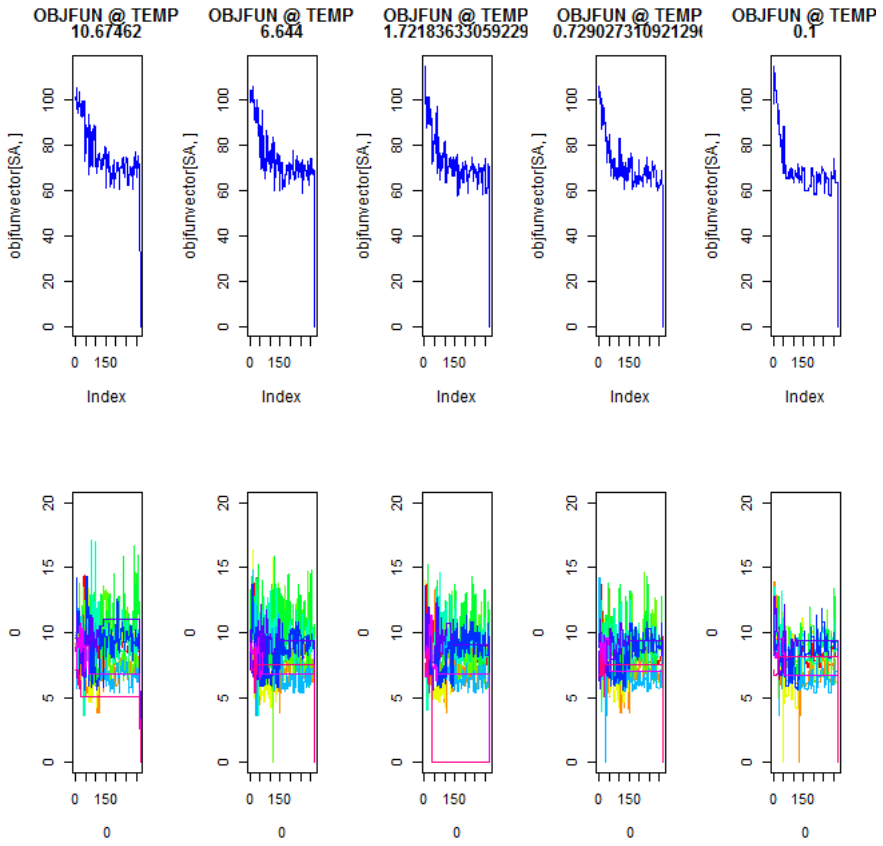
6.3 Parallel Tempering

The key parameter in PT is the number of parallel chains and the temperature of these chains. The hottest temperature must be hot enough such that it is capable of exploring wide portions of the search space, while the coolest state should only do local search in the neighborhood of a good solution and only accept improvements. Also, the temperatures should be close enough to

each other such that the state switch moves have a decent probability of being accepted.

Parallel Tempering works best is there are many local minima close to each other.

Figure 9: Typical behaviour of Parallel Tempering. Note that the higher temperature states have higher variance in the objective function. Also note that most of the time solutions produced by PT are not feasible because of individual pathlengths. Image produced by R.



A good guideline for temperature levels is that

$$\left(\frac{1}{T_i} - \frac{1}{T_{i+1}}\right)|\delta U| \approx -\log p_a$$

where $|U|$ is the mean energy change under the target distribution. In our

case this is around 10% of the total cost, therefore between 5 and 8.

Also, p_a is the lower bound for the acceptance ratio which is close to $\frac{1}{e}$, so we can treat the right hand side as 1.

Together this means that in our computation

$$\left(\frac{1}{T_i} - \frac{1}{T_{i+1}}\right) \approx \frac{1}{6}$$

I used a different strategy to calculate temperatures as explained in the implementation chapter.

In the experiment I used 5 different temperature states based on preliminary tests. The algorithm needs much more in theory, but The biggest constraint was computation time. In my implementation running 5 chains PT with 500 steps each took around 10 times more time then running SA with 2500 steps. This is because PT uses much more computer memory than SA. Therefore I had to use much less total iterations in PT then in SA. Surprisingly, the results showed that PT still had a better performance in general.

For tuning the temperatures I used an initial scheme. However, the algorithm monitored the relative variance of the objective function in each state and each state had a target variance. I based these target variances on the intended purpose of each state. The highest temperature should make big jumps and explore wide areas of the search space, while the coolest temperature should capture the best solutions and do local search on them. If the variance was higher then the target variance for a state then the temperature was decreased and vice versa. This way the algorithm found its own temperature schedule.

7 Practical results

In this section I describe some case studies in which the authors applied computational techniques to reduce collection routes of trash hauling companies. I chose trash hauling as a topic because I was very interested in the way thrash collection works in Budapest. These case studies illustrate the possible gains from using these methods.

7.1 Nikea, Athens 2010

The local municipality bought route optimization services from a private company. The procedure had 2 parts. In the first part, they reduced the number of

trash bins by using bigger, uniform sized bins and they put them in more easily accessible places. In this way they reduced the number of collection points by more than 60%. The second step was using algorithms to make a new route. They used the so-called "empirical solution" as an initial solution. The empirical solution is the route previously used by truck drivers.

Interestingly, in this case most of the gains resulted from the first step, and not the second. This points to the fact that algorithms are not everything in real world applications.

The total length of routes decreased by 12.5% while the time needed for the collection decreasd by 17

7.2 Ipoh City, Malaysia 2014

Ipoh City is a city with population 700 thousand. In this study the authors collaborated with the local municipality and selected a district for the optimization. The article stresses the fact that 50-70% of the costs of waste management is associated with collection.

As a result of the project they reduced the length of collection routes by 20%.(26)

7.3 Onitsha, Nigeria 2008

Onitsha has around 500 thousand residents. Before implementing the planning system, the drivers had a free hand in making their collection routes. The project resulted in a significant cost reduction, but this is mainly because the poor quality of the original solution.

In total, they reduced the total length of the routes by 16% and the costs associated with waste management were reduced by 25%.(27)

7.4 Trabzon, Turkey 2007

Trabzon is a 170 thousand resident city in the north of Turkey. The researchers used on board video cameras on the trucks to get a better picture of the true costs associated with waste collection. On average the length of the routes were reduced by 27%, the time needed was reduced by 44%. All in all, the costs were decreased by 25%. (28)

8 Experiment design

I conducted 3 experiments to compare the performance of my implementation of the algorithms. I coded all algorithms in R from complete scratch and wrote around 10000 lines of code. I plan to publish them as an open access R package later.

In each experiment I created 10 random examples of 90 points + 1 Depot each, and tried to solve the capacity and lengthconstrained VRP. The length-constraint was always 10, and the capacityconstraint was 8. All graph instances were undirected and the adjacency matrix was symmetric. The 3 experiments had the following properties:

1. Experiment: The points were somewhat clustered, but the clusters were not easily identifiable. The capacityconstraint was effective. The Depot was in the top right corner. This is important for the quality of initial solutions. The MCMC methods could create paths that were 10% above the capacity constraint

2. Experiment: The points were in 9 easily identifiable clusters, but the capacity constraint forbid that each cluster was served by 1 route. The capacity of each point was uniformly distributed in $[0, 1]$, the capacity constraint was 6 and there were 10 points in each cluster. The MCMC methods were not allowed to create routes above the capacity constraint.

3. Experiment: The points were all in 1 big cluster. The Depot is in the top right corner. The MCMC methods were not allowed to create routes above the capacity constraint.

I produced 3 initial solutions from 2 versions of the CW algorithm and the Sweep algorithm. I picked the best (usually Sweep) and fed it to the more sophisticated ones. I allowed them to run for a number of iterations such that the running times were in the same order of magnitude. I would like to stress that I went for equal running times and not equal number of iterations. I used the US procedure on all the final solutions to improve the objective function.

In the end I recorded the running time, the objective function at the best feasible solution, the number of routes and the utilization of capacity and route-length. The last 2 parameters measure if the routes were "full" or not.

My intial goal was to explore if different algorithms are needed to solve different instances of the same VRP problem. I wanted to discover which algorithms

work best in which type of problems. The 3 experiments are mostly different in one particular way: the distance between feasible solutions in the search space. In experiment 2 there are clusters of neighbouring feasible solutions (for example when each cluster is serviced by 2 paths), while in Experiment 3 the neighbouring feasible solutions are uniformly scattered in one big region. This is because the points are in the same direction relative to the Depot and they are relatively close to each other. Experiment 1 is in between these 2, I used it primarily to fin tune the algorithms.

9 Experiment results

9.1 Experiment 1

In experiment 1 the Tabu Search and the MCMC methods had just about the same results. The results are summarized in the next table. It is obvious that there is no best algorithm in this experiment. The differences between the performances are not significant. Tabu Search got stuck in the beginning in a few instances. The running time of Simulated Annealing is significantly lower despite the fact that Paralell Tempering did much less iterations overall. It is also noteworthy that Parallel Tempering produced on average 0.6 less paths than Tabu Search.

The best algorithm could reduce pathlengths by 36% on average. The post optimization procedure improved SA by 14% and did not significantly improve PT.

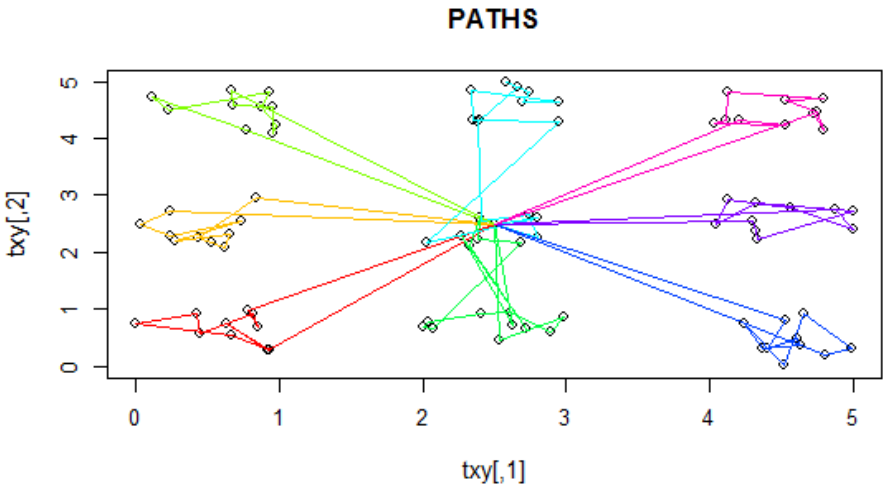
Figure 10: Results of the 1. Experiment.

Problem Instance	BEST SOLUTION	BEST ALG	SWEEP	Reduction from SWEEP	Length of best solution			Number of routes			Running Time (seconds)		
					SA US	PT US	TABU SEARCH	SA US	PT US	TABU SEARCH	SA	PT	TABU SEARCH
1	46	TABU SEARCH	74.22	62%	51.1	51	46	8	8	8	201	632	1302
2	50	TABU SEARCH	73.24	68%	55.7	51	50	8	8	8	186	686	988
3	50	PT US	80.91	62%	61.3	50	54	9	8	9	206	698	900
4	46	SA US	75.46	61%	45.8	49	64	7	8	9	186	586	25
5	48	PT US	75.88	63%	57.0	48	50	9	8	9	236	879	1076
6	46	TABU SEARCH	69.49	66%	55.7	46	46	8	8	8	170	697	715
7	55	SA US	85.24	64%	54.6	55	56	9	8	9	230	759	1214
8	47	TABU SEARCH	72.14	65%	52.4	49	47	8	8	8	213	676	726
9	47	TABU SEARCH	72.07	65%	50.7	48	47	8	8	8	197	738	1353
10	48	PT US	75.29	63%	49.2	48	53	9	7	9	165	712	389

9.2 Experiment 2

In this experiment the MCMC methods clearly outperformed Tabu Search. There are 2 reasons for that. First, Tabu Search got stuck in 3 instances and could not find a significantly better version that Sweep. On the other hand, the MCMC methods were much more effective in reducing the number of paths, especially Parallel Tempering. The best algorithm could improve 42% on the original solution. It is also important to note that while PT outperformed SA without postoptimization by 10%, after applying US to the best feasible solution this difference shrunk to 2%. A typical problem instance of experiment 2 is on the next figure.

Figure 11: Experiment 2 instance



And the next table presents a summary of results of Experiment 2.

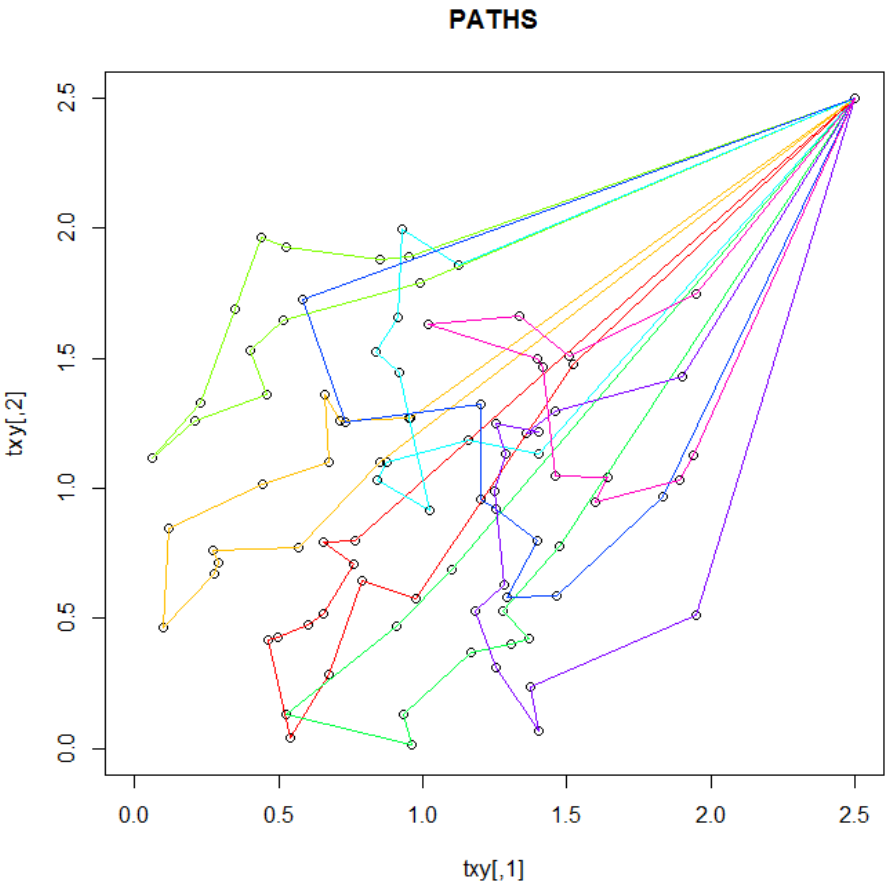
Figure 12: Results of the 2. Experiment.

Problem Instance	BEST SOLUTION	BEST ALG	SWEEP	Reduction from SWEEP	Length of best solution			Number of routes			Running Time (seconds)		
					SA US	PT US	TABU SEARCH	SA US	PT US	TABU SEARCH	SA	PT	TABU SEARCH
1	57	SA US	90.02	63%	56.8	57	88	8	8	10	141	495	11
2	60	PT US	96.95	62%	59.7	60	91	9	9	11	128	463	119
3	59	PT US	104.17	57%	61.1	59	79	9	9	11	125	565	431
4	57	PT US	96.94	58%	57.1	57	74	9	9	11	142	507	753
5	56	PT US	113.41	50%	59.2	56	111	8	8	13	155	592	11
6	57	PT US	100.59	57%	58.3	57	80	8	8	11	133	555	903
7	57	PT US	105.74	54%	59.1	57	103	8	8	12	140	548	17
8	59	PT US	100.83	59%	64.0	59	81	9	9	11	154	673	721
9	59	PT US	90.80	65%	59.3	59	69	9	9	10	147	600	707
10	58	PT US	96.84	60%	58.2	58	86	8	9	11	168	590	473

9.3 Experiment 3

In this experiment it is unclear which algorithm was the best, however Tabu Search had the best performance on average, but the difference is not significant. What is more interesting, is that SA did better than PT. The average reduction of the objective function was 37%, quite similar to Experiment 1. On the other hand, the average number of paths constructed is higher compared to Experiment 1.

Figure 13: Experiment 3 instance



The results of Experiment 3 are summarized in the following table

Figure 14: Results of the 3. Experiment.

Problem Instance	BEST SOLUTION	BEST ALG	SWEEP	Reduction from SWEEP	Length of best solution			Number of routes			Running Time (seconds)		
					SA US	PT US	TABU SEARCH	SA US	PT US	TABU SEARCH	SA	PT	TABU SEARCH
1	48	TABU SEARCH	74.10	65%	49.9	51	48	8	8	8	296	616	853
2	50	SA US	81.65	62%	50.4	56	53	8	9	9	352	661	1076
3	48	SA US	92.64	52%	48.3	58	60	7	9	10	201	655	1221
4	51	TABU SEARCH	73.53	69%	52.6	52	51	8	8	8	166	769	801
5	52	TABU SEARCH	82.09	63%	52.3	52	52	8	8	9	176	735	1970
6	49	TABU SEARCH	76.30	64%	55.3	56	49	8	8	8	124	426	1322
7	50	SA US	83.53	60%	49.9	54	55	7	8	9	178	667	1234
8	45	PT US	75.32	60%	52.6	45	51	8	7	8	266	655	1173
9	52	PT US	79.08	66%	55.3	52	53	8	8	9	183	660	1089
10	50	TABU SEARCH	76.29	65%	57.9	59	50	8	8	8	114	387	1062

10 Conclusion

I presented a wide review of literature in the field of Vehicle Routing Problem optimization, especially MCMC methods, and proved that the VRP is an NP-hard combinatorial optimization problem. After that I presented 6 implementations of the described algorithms in R and conducted 3 different experiments with them.

My initial hypothesis was that on structurally different examples the best algorithms would be different. In the case of Experiment 2 I could confirm my hypothesis, as Tabu Search was clearly inferior to the MCMC methods.

Possible future directions for my research could include implementing these algorithms to solve problems with asymmetrical distance matrices on directed graphs, possibly such that the triangle inequality is not always satisfied.

This is an important direction of research, as most cities (including Budapest) have one-way streets and bridges and the symmetry of the distance matrix is not guaranteed. This way, I can solve truly real world instances of the problem.

References

[1] N. Biggs, *Algebraic Graph Theory*, 2nd ed. Cambridge: Cambridge University Press, n.a.

[2] E. W. Weisstein. (n.a) Graph cycle. [Online]. Available: http://mathworld.wolfram.com/GraphCycle.html

[3] M. Barile and E. W. Weisstein. (1999) Path. [Online]. Available: http://mathworld.wolfram.com/Path.html

[4] E. W. Weisstein. (n.a) Np problem. [Online]. Available: http://mathworld.wolfram.com/NP-Problem.html

[5] ——. (n.a) Np-hard problem. [Online]. Available: http://mathworld.wolfram.com/NP-HardProblem.html

[6] ——. (n.a) Np-complete problem. [Online]. Available: http://mathworld.wolfram.com/NP-CompleteProblem.html

[7] B. Esfahbod, "P and np," 2007. [Online]. Available: https://commons.wikimedia.org/wiki/File:P_np_np-complete_np-hard.svg

[8] M. Barile and E. W. Weisstein. (1999) Hamiltonian cycle. [Online]. Available: http://mathworld.wolfram.com/HamiltonianCycle.html

[9] G. Pataki, "Teaching integer programming formulations using the traveling salesman problem," *SIAM Review*, vol. 45, no. 1, pp. 116–123, 2003.

[10] G. Dantzig, R. Fulkerson, and S. Johnson, "Solution of a large-scale traveling-salesman problem," *Operations Research*, vol. 2, pp. 393–410, 1954.

[11] G. Carpaneto, S. Martello, and P. Toth, "Algorithms and codes for the assignment problem," *Annals of Operations Research*, vol. 13, no. 1, pp. 191–223, 1988. [Online]. Available: http://dx.doi.org/10.1007/BF02288323

[12] D. L. Applegate, R. E. Bixby, V. Chvatal, and W. J. Cook, *The Traveling Salesman Problem: A Computational Study (Princeton Series in Applied Mathematics)*. Princeton, NJ, USA: Princeton University Press, 2007.

[13] G. Woeginger, "Exact algorithms for np-hard problems: A survey," *Combinatorial Optimization – Eureka, You Shrink! Lecture notes in computer science*, vol. 2570, pp. 185–207, 2003.

[14] J. B. Kruskal, "On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem," *Proceedings of the American Mathematical Society*, vol. 7, no. 1, pp. 48–50, Feb. 1956. [Online]. Available: http://www.jstor.org/stable/2033241

[15] M. Gendreau, A. Hertz, and G. Laporte, "New insertion and postoptimization procedures for the traveling salesman problem," *Oper. Res.*, vol. 40, no. 6, pp. 1086–1094, Nov. 1992. [Online]. Available: http://dx.doi.org/10.1287/opre.40.6.1086

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

[17] J. Lysgaard, "Athe clark and wright's savings algorithm," 1997.

[18] A. Wren and A. Holliday, "Computer scheduling of vehicles from one or more depots to a number of delivery points," *Operations Research Quarterl*, vol. 23, pp. 333–344, 1972.

[19] M. Gendreau, A. Hertz, and G. Laporte, "A tabu search heuristic for the vehicle routing problem," *Management Science*, vol. 40, no. 10, pp. 1276–1290, 1994. [Online]. Available: http://www.jstor.org/stable/2661622

[20] S. Chib and E. Greenberg, "Understanding the Metropolis-Hastings Algorithm," *The American Statistician*, vol. 49, no. 4, pp. 327–335, Nov. 1995. [Online]. Available: http://dx.doi.org/10.2307/2684568

[21] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller, "Equation of state calculations by fast computing machines," *The Journal of Chemical Physics*, vol. 21, no. 6, pp. 1087–1092, 1953. [Online]. Available: http://link.aip.org/link/?JCP/21/1087/1

[22] W. K. Hastings, "Monte Carlo sampling methods using Markov chains and their applications," *Biometrika*, vol. 57, no. 1, pp. 97–109, Apr. 1970. [Online]. Available: http://dx.doi.org/10.1093/biomet/57.1.97

[23] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *SCIENCE*, vol. 220, no. 4598, pp. 671–680, 1983.

[24] I. H. Osman, "Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem," *Annals of Operations Research*, vol. 41, no. 4, pp. 421–451, 1993. [Online]. Available: http://dx.doi.org/10.1007/BF02023004

[25] C. J. Geyer, "Markov chain monte carlo maximum likelihood," pp. 156–163, 1991.

[26] A. Malakahmad, P. M. Bakri, M. R. M. Mokhtar, and N. Khalil, "Solid waste collection routes optimization via gis techniques in ipoh city, malaysia," *Procedia Engineering*, vol. 77, no. Complete, pp. 20–27, 2014.

[27] T. Ogwueleka, "Route optimization for solid waste collection: Onitsha, nigeria case study," *Journal of Applied Sciences and Environmental Management*, vol. 13(2), no. Complete, pp. 37–40, 2009.

[28] O. Apaydin and M. Gonullu, "Route optimization for solid waste collection: Trabzon, turkey case study," *Global NEST Journal*, vol. 9, no. Complete, pp. 6–11, 2007.