Application Framework for Snow Removal Routing Problem
by
Sajjadul Islam
LIU-IDA/LITH-EX-A--10/012--SE
2010-03-03
Final Thesis

Application Framework for Snow Removal Routing Problem

by

Sajjadul Islam

LIU-IDA/LITH-EX-A--10/012--SE

2010-03-03

Supervisor: Kristian Sandahl, IDA
Gholamreza Razmara, Optiplan AB
Examiner: Kristian Sandahl, IDA
Application Framework for Snow Removal Routing Problem

by
Sajjadul Islam

A Thesis submitted in partial fulfillment of requirements for the degree of Master of Science in Information Technology
Linköping University
2010
Abstract

Road maintenance during the heavy snowfall is an important problem. In Sweden the government and municipalities together spend close to 0.3 thousand million SEK every year for winter road maintenance. Approximately half of it is snow removal cost, which in turn to a large extent depends on the routing of the snow-ploughing vehicles. In this thesis work, we wish to develop an application framework for optimized routing operation for these vehicles so that the total operational cost can be reduced at a significant level. In general, there are different characteristics of snow ploughing depending on the routing procedure, one is done after the snowfall and the other is during the snowfall. In this thesis work, we are only interested to find the set of routing paths during the snowfall where duration of snowfall is unknown. We present a new way of generating an initial solution that deals with the real operational network. The optimization algorithm works upon this initial solution and try to reduce the number of periodic paths.
Acknowledgments

I would like to thank my supervisor Kristian Sandahl at IDA for ideas and feedback for putting this thesis together. It could not have come out without the support of Optiplan AB’s continuous support, specially Gholamreza Razmara, Optimization Expert & System Developer and Jonas Åhlin, Chief Executive Officer. Also thanks to Tommy Olsson at IDA for providing useful suggestion in faster table-look-up concepts.
Contents

Abstract i

Acknowledgements iii

List of Figures vii

List of Tables ix

1 Introduction 1
   1.1 Problem formulation .................................................. 2
   1.2 Industrial Partners .................................................... 2
   1.3 Research method ...................................................... 2
   1.4 Outline ................................................................. 3
   1.5 Prerequisite ............................................................ 3
   1.6 Terminology ............................................................. 3

2 Theoretical Background 5
   2.1 Snow Removal and Winter Road Maintenance .......................... 5
   2.2 Snow Removal Problems ............................................... 7
   2.3 Vehicle Routing Problems ............................................. 8
   2.4 Various types of Snow Removal Routing Problem ..................... 9
   2.5 The Networks .......................................................... 9
      2.5.1 The Road Network of Eskilstuna .................................. 9
      2.5.2 During Snowfall ................................................... 11
      2.5.3 Applied Optimization and Operations Research ................. 11

3 Mathematical Formulation 15
   3.1 Overview and solution approaches ................................... 15
   3.2 Time Periodic Formulation of SRRPTWDS Problem .................... 17
   3.3 Simulated Annealing .................................................. 17
      3.3.1 The method ...................................................... 18
      3.3.2 Comparison with other methods ................................ 20
4 Initial Solution Generation

4.1 Problem Formulation ........................................ 21
4.2 An Initial Solution ........................................... 22
4.3 Periodic Route Path Segment Removal: ......................... 27
4.4 Periodic Route Path Segment Insertion: ....................... 27
4.5 Swap Operation of Periodic Route Path Segments .............. 30
4.6 Split Operation Periodic Route Path Segments: ............... 33
4.7 Merge Operation of Periodic Route Path Segments ............ 33
4.8 Cleanup Operation of Redundant Transport Paths ............. 34
4.9 Algorithm work-flow ........................................ 36

5 Implementation and Results .................................. 37

5.1 Software requirement analysis ................................ 38
5.1.1 Functional requirement ..................................... 38
5.1.1.1 Specific requirements .................................. 38
5.1.2 Nonfunctional requirement ................................. 39
5.2 Requirements elicitation ....................................... 39
5.2.1 Requirement sources ....................................... 39
5.2.2 Elicitation techniques ..................................... 39
5.2.3 Requirements classification ................................. 39
5.3 Software design .............................................. 39
5.3.1 Software architectural design .............................. 39
5.3.2 Software detailed design .................................. 40
5.3.2.1 Class Design ......................................... 40
5.4 Testing ....................................................... 48
5.5 Exception Handling ........................................... 48
5.6 Timeframe .................................................... 48
5.7 Results ....................................................... 49

6 Discussion .................................................... 51

6.1 Learning Experience .......................................... 51
6.1.1 Area of choice ............................................. 51
6.1.2 Applied optimization concept ............................... 51
6.1.3 Data-structure and algorithm ............................... 52
6.1.4 Adhoc testing procedure .................................. 52
6.2 Future steps .................................................. 52
6.2.1 Improved Object-Oriented Generic Programming .......... 52
6.2.2 Parallel Programming ...................................... 53

7 Conclusion and Future Work .................................. 55
List of Figures

2.1 Winter maintenance cost of national road network over the duration of five years . 5
2.2 Winter road maintenance cost distribution ................................. 6
2.3 Operation district of Eskilstuna ............................................. 10
2.4 Edge representation in mathematical model ............................. 11
3.1 The structure of Simulated Annealing algorithm ...................... 19
4.1 Periodic route formation procedure ......................................... 21
4.2 Initial solution generation .................................................. 24
4.3 Generation of Initial Solution(initial version) ......................... 25
4.4 Expanded activity diagram of Initial Solution Generation .......... 25
4.5 Neighboring edge choosing criteria ....................................... 26
4.6 Neighboring edge choosing criteria based on shortest path .......... 26
4.7 Path segment removal from Periodic Route ......................... 27
4.8 Insertion of a two-way directed *guest* path segment into two-way directed *host* path segments of a periodic route ................................................ 28
4.9 Insertion of one-way directed *guest* path segment into two-way directed *host* path segments of periodic route .............................. 29
4.10 Insertion of two-way directed *guest* path segment into 1-way directed *host* path segments of periodic route ......................... 29
4.11 Insertion of a one-way directed *guest* path segment into one-way directed *host* path segments in periodic route .............................. 30
4.12 Insertion of one-way directed *guest* path segment into mixed way directed *host* path segments of periodic route ......................... 31
4.13 Insertion of 2-way directed *guest* path segment into mixed way directed *host* path segments of periodic route .............................. 31
4.14 Cleanup operation while insertion ....................................... 32
4.15 A container with plough path segment and similar time window .... 34
4.16 Removal of 1-way directed transportation paths at the beginning of the periodic route paths ....................................................... 35
4.17 Removal of 1-way directed transportation paths at the trailing of periodic route paths .............................. 35
4.18 Simulated Annealing activity diagram .................................. 36
5.1 Evolution-tree life-cycle model ............................................ 37
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.2</td>
<td>functional requirement of the system</td>
<td>38</td>
</tr>
<tr>
<td>5.3</td>
<td>High-level architectural diagram</td>
<td>40</td>
</tr>
<tr>
<td>5.4</td>
<td>Network representation by means of Adjacency set</td>
<td>41</td>
</tr>
<tr>
<td>5.5</td>
<td>Node Class Diagram</td>
<td>42</td>
</tr>
<tr>
<td>5.6</td>
<td>Node object with reference to the Map location and the Adjacency set</td>
<td>42</td>
</tr>
<tr>
<td>5.7</td>
<td>Edge class diagram</td>
<td>43</td>
</tr>
<tr>
<td>5.8</td>
<td>Edge object with reference to the Map</td>
<td>43</td>
</tr>
<tr>
<td>5.9</td>
<td>Vehicle class diagram</td>
<td>44</td>
</tr>
<tr>
<td>5.10</td>
<td>Vehicle Map and Vector networkVehicleList</td>
<td>45</td>
</tr>
<tr>
<td>5.11</td>
<td>Network class</td>
<td>46</td>
</tr>
<tr>
<td>5.12</td>
<td>Neighbor class diagram</td>
<td>46</td>
</tr>
<tr>
<td>5.13</td>
<td>VehicleEdgeConnectivity class</td>
<td>46</td>
</tr>
<tr>
<td>5.14</td>
<td>ShortestPathAttributesBetweenNodes class</td>
<td>47</td>
</tr>
<tr>
<td>5.15</td>
<td>PeriodRoute class</td>
<td>47</td>
</tr>
<tr>
<td>5.16</td>
<td>Solution class</td>
<td>48</td>
</tr>
<tr>
<td>5.17</td>
<td>Sioux Falls Network representation</td>
<td>49</td>
</tr>
<tr>
<td>5.18</td>
<td>Period Route Representation of Sioux Falls</td>
<td>50</td>
</tr>
</tbody>
</table>
List of Tables

1.1 Abbreviations used in the thesis ........................................ 4

2.1 Classification of roads segments (traffic lanes) by SNRA standards. Time periods are
given in hours ........................................................................ 6

3.2 Comparison table between Neural sets and SA ........................... 20
3.3 Comparison table between Genetic algorithm and SA .................... 20
Chapter 1

Introduction

One of the main tasks of road administration during winter season in snow falling countries is to spread salt, grit, remove snow and dispose them. Salt spreading/gritting is the activities of spreading salt before the roads become very slippery due to snowing. By snow removal we mean ploughing of snow from roads while snow disposal consists of ploughing, as well as loading and transporting them to the disposal sites.

In countries where snow create major problems and where heavy snowfalls may result in major traffic accidents, disorder and disrupting the traffic flow, the Road Maintenance Administration attempts to keep the roads operational and safe by large expensive programs for the task of snow removals. Due to the heavy cost of of snow removal, it becomes an important issue to reduce the cost by optimizing the routing operation. We try to find a set of routes with associated time schedules for a number of available snowploughs\(^1\) that minimize the total cost of the snow removal operation. In practice, the routing of snowploughs is performed by experts with good knowledge of the underlying road network. Once the route is generated using one of the optimization algorithms, it can be used for several years with some minor modification within the underlying network, if necessary. This motivates us to find a solution to route planning of snowploughs in a more systematic way where we have considered both single and both way directed edges in the network to generate a solution and executed the optimization algorithm to achieve a better solution. We have tried to follow object-oriented design with efficient data-structure to begin developing this framework. The object-oriented paradigm has many strengths that motivates us to follow it. They are:

- The object-oriented paradigm supports information hiding. Consequently, if implementation details are changed within an object during the maintenance, information hiding ensures that no other part of the system need to be modified to ensure consistency. So the object-oriented paradigm makes maintenance quicker and easier and the chance of introducing regression fault is reduced.

- Along with the maintenance, the object-oriented paradigm makes development easier. Modeling plays a major role in the paradigm. The close correspondence between the product and their counterparts in the real world should lead to a better quality software.

- Well-designed objects are independent units. An object encompasses both attributes and operations performed on the attributes. If all the operations performed on the attributes of an object are included on that object, the object can be considered a conceptually independent entity.[Stephen2008]

- Object-oriented paradigm promotes reuse. Since objects are independent entities, they can be utilized in future products. This reuse of objects reduce the time and cost of both development and maintenance.

\(^1\)snow ploughing vehicles
1.1 Problem formulation

The question this thesis want to answer is:

- How can we generate an optimal set of periodic routes in network for snowploughs?
- How can we generate a solution in a reasonable time?
- What will be quality of the solution?
- How can we improve the design of the algorithm?
- How can we even say that the solution that we have found so far is the optimal one or even near to optimal solution?

This problem can be divided into sub-problems. And according to well-known journalist’s question of who, what, when, where, why and how will be formulated for this problem[RL1995]. Some of the questions will be answered directly, but some will remain for further investigation.

- Who will use that application framework to generate paths to plough and salt?
- What optimization process will be used to provide that solution?
- How optimal the solution is than the current one?

Trying to answer these questions, we get:

- The road transport authority( SNRA ) who are responsible for the snow-ploughing.
- The simulated annealing algorithm will be used to provide the solution.
- There no guarantee that the solution is the optimal one, but we can guarantee that it will be as good as the current solution.

1.2 Industrial Partners

This thesis work is carried out in cooperation with IDA(The Department of Computer and Information Science), Linköping University and Optiplan AB, located at PRONOVA Science Park in Norrköping.

1.3 Research method

The research method on software engineering is often an ad hoc process that is seldom addressed in any software engineering literature[RL1995]. To address the need for structured research in software engineering Glass proposes the separation of research in the following four phases:

- The information phase - All the relevant information is gathered via reflection, literature study or a poll.
- The propositional phase - A hypothesis, theory or model is proposed.
- The analytical phase - The stated proposition is analyzed, leading to a formulation of a theory.
- The evaluation phase - The proposition or the analytical findings are evaluated by means of experimentation or observation.

We have planned to use the above framework for this thesis work.
1.4 Outline

Chapter 1: Introduction This chapter contains the problem formulation, research method, and other introductory information.

Chapter 2: Theoretical Background The purpose of this chapter is to provide the reader with a framework in which the thesis can be studied. The concepts of winter road maintenance, the road network, different types of snow removal problems and the operational district under consideration is outlined.

Chapter 3: Mathematical Formulation This chapter provides with a mathematical description of snow removal routing problem after and during snowfall. The description is followed by a survey on the core algorithm - Simulated Annealing that had been used extensively to solve other routing problems in the past and motivation to use it in periodic SRRPTWDS as well.

Chapter 4: Initial Solution Generation This chapter contains not only the description of the algorithm that is used to generate an initial solution for periodic SRRPTWDS, but also discusses several other algorithms that are used by simulated annealing to come up with an optimized solution.

Chapter 5: Implementation and Results This chapter briefly outline the requirement analysis and implementation along with UML activity and class diagrams of the ideas presented in the previous chapter.

Chapter 6: Discussion This chapter describes the overall experiences of this thesis work. It assesses the positive and negative feeling while going through different phases of the work. It also analysis in relation to the original aims and objectives. And it concludes with suggestion of new approaches that might be used for re-engineering.

Chapter 7: Conclusions and Future Work The last chapter summarizes the thesis and discusses possible extensions of this work.

1.5 Prerequisite

The reader is assumed to have basic understanding of software development, data-structure, and optimization theory. The Implementation and Results chapter will be inaccessible to the reader without some basic concept of STL in C++.

1.6 Terminology

SRRPTWDS is a general term for Snow Removal Routing Problem With Time-window During Snowfall which is used all through this report. There are some other abbreviations which have been elaborated in table 1.1:
### Abbreviations

<table>
<thead>
<tr>
<th>Abbreviations</th>
<th>Signifies</th>
</tr>
</thead>
<tbody>
<tr>
<td>SRP</td>
<td>Snow Removal Problem</td>
</tr>
<tr>
<td>SRRPTWDS</td>
<td>Snow Removal Routing Problem With Time-window During Snowfall</td>
</tr>
<tr>
<td>SRRPTWAS</td>
<td>Snow Removal Routing Problem With Time-window After Snowfall</td>
</tr>
<tr>
<td>SNRA</td>
<td>Swedish National Road Administration</td>
</tr>
<tr>
<td>SRRPAS</td>
<td>Snow Removal Routing Problem After Snowfall</td>
</tr>
<tr>
<td>CSPP</td>
<td>Constrained Set Partitioning Problem</td>
</tr>
<tr>
<td>SPP</td>
<td>Shortest Path Problem</td>
</tr>
<tr>
<td>VRP</td>
<td>Vehicle Routing Problem</td>
</tr>
<tr>
<td>SA</td>
<td>Simulated Annealing</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphical User Interface</td>
</tr>
<tr>
<td>UML</td>
<td>Unified Modeling Language</td>
</tr>
<tr>
<td>ODE</td>
<td>Operational District of Eskilstuna</td>
</tr>
<tr>
<td>STL</td>
<td>Standard Template Library</td>
</tr>
</tbody>
</table>

Table 1.1: Abbreviations used in the thesis
Chapter 2

Theoretical Background

This chapter is basically the information gathering phase where we do some literature study to formulate the concept of snow removal routing problem in general and then focus specifically with a type of snow removal problem. We start with the estimated cost that SNRA spend each year to remove snow from different road types irrespective of the time instant chosen (after snowfall or during the snowfall). This is followed by some background work that has been done in the domain of snow removal problem and specially vehicle routing problems over years. Then we give a description of our operational network on which we shall run our algorithm. We end the chapter with the overview of the optimization algorithm that is used in operations research.

2.1 Snow Removal and Winter Road Maintenance

In most heavy snow falling countries, snow and ice on the roads lead to considerable maintenance costs and socioeconomic consequences such as increased accident risks, reduced accessibility, increased fuel consumptions and vehicle corrosion costs. The maintenance of roads done by SNRA, i.e. salt spreading, gritting, and snow removal, including planning, purchase of equipment etc generates costs for the government up to SEK 1.75 billion and maintenance of the municipal roads about 1 billion every year of which the half corresponds to the snow removal of roads [Golam2004]. Figure 2.1 compares the total costs of winter road maintenance for SNRA, for a period of five years [Golam2004].

![Figure 2.1: Winter maintenance cost of national road network over the duration of five years](image)

Figure 2.1: Winter maintenance cost of national road network over the duration of five years

The distribution of the annual road maintenance cost over different types of operations for winter road maintenance of the national road network is depicted by the figure 2.2 [Golam2004]:
CHAPTER 2. THEORETICAL BACKGROUND

According to Gholamreza(2004, pp. 5), to guide the winter road maintenance, operative standards for road conditions services are set forth by the SNRA. These standards describes in detail the demands on the road condition services for snow and skid-free roads as well as snow-covered roads. Gholamreza(2004, pp. 5) specified that a snow and skid-free road is defined as a road normally free from snow and ice, and if possible dry in fair weather within a certain time after precipitation. A snow-covered roadway is the one that is packed of ice and snow during winter. In fair weather within a certain time after precipitation, the surface must be even and free of loose snow, in addition to having satisfactory friction over the surface. These standards are described in a technical publication of the SNRA, Operation 96\(^1\), and the modified version, WINTER 2003\(^2\), which are the general technical descriptions of operation service levels. According to Operation 96, snow and skid-free roads are divided into four standard classes. A comparison of classifications is made in table 2.1 [Golam2004].

<table>
<thead>
<tr>
<th>Time period within a road segment must be ploughed</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standard class according to Operation 96</td>
<td>A1</td>
<td>-</td>
<td>A2</td>
<td>-</td>
<td>A3,A4,B1</td>
<td>B2</td>
</tr>
<tr>
<td>Standard class according to WINTER 2003</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 2.1: Classification of roads segments (traffic lanes) by SNRA standards. Time periods are given in hours

From table 2.1 we can categorize different classes of roads as follows:

- **A1** - This class of road have a period of 2 hours, which means that these must be ploughed every 2 hours during snowfall.
- **A2** - This class of road must be ploughed every 4 hours during snowfall.
- **A3,A4,B1** - These class of roads must be ploughed every 5 hours during snowfall.
- **B2** - This class of road segment must be ploughed every 8 hours during snowfall.

Due to the large cost of snow removal, decreasing the total cost becomes an important issue. It can be achieved by undertaking different measures such as:

- Determine the optimal distribution and allocation of resources and equipments.
- Determine the optimal set of routes for snowploughs.
- Determine the optimal crew scheduling program.

Here we try to generate an optimal set of routes for the available snow ploughing vehicles.

\(^1\)General technical description of operation service level published in 1996 by SNRA.
\(^2\)General technical description of operation service level published in 2003 by SNRA.
2.2 Snow Removal Problems

In spite of the importance of snow removal and disposal, gritting and salt spreading, the amount of scientific work on these subjects are limited. These problems have not been extensively studied even though the annual expenditures in countries where the problems mentioned here are major issues. In this section an overview of some existing research on SRP is given and an effort is made to introduce various presented solution approaches to these problems.

In the early 70's, Mark and Stricker (1971) [HR1971] focused on routing of public service vehicles and in particular discussed the snow removal and garbage collection problems. The main distinction between these two problems lies in priority constraints in SRP. Since in general a single vehicle cannot cover the entire road network they suggest solving the incapacitated version of garbage collection by solving Mixed-Chinese Postman Problem, M-CPP, and explain the similarity between these problems. The Chinese Postman Problem is to find the shortest route path in a network that uses every directed edge and get back to where they started (closed problem) from or does not get back (open problem).

In Liebling (1973)[TM1973], the routing of snow-ploughs is tackled by a two-phase procedure, a rough and fine planning phase. In fine-planning phase, we find a traversal of minimal length that covers a given subset of a connected segments of a city network. And in the rough planning phase, the road network is divided into a number of disjoint sections depending on vehicle capacity, and some other relevant attributes of the crews.

The problem of routing the salt spreader vehicles in an urban environment based on the capacity restrictions of the vehicles is studied by Cook and Alprin(1976)[MS1976]. The objective was to minimize the time required to spread salt over a given road segment. The framework presented by Minsk(1979)[LD1979] for a system analysis of snow removal consists of the following four steps:

- Operating conditions - Climate, road, and traffic.
- System - Traffic of the road.
- System objectives - Keep the network operational during and after periods of snow at minimum cost.
- Control measures - Mechanical/chemical removal, application of abrasives, traction aides on vehicles, closing of routes.

He described each component of the system along with their associated effects on each other and analyze the equipment factors involved in performing the basic functions of snow removal.

Tucker and Cohan(1980)[WG1980] generated a general computer model to simulate urban SRP. The simulation package assists the end user in determining the routing the snow-ploughs interactively by using the computer graphics visualization techniques and once the routes are input, any particular snow removal can be simulated by changing a set of parameters including vehicle and snow storm characteristics. They computed the time period to plough each road segment and the minimum ploughing time for a given fleet (vehicle) to clear the city. The end user can then determine the minimum ploughing time for each vehicle. To assess the output, they applied the simulation program to several cities and reported encouraging results. They believed that the simulation can be used to increase the efficiency of various routing strategies.

Lemieux and Campagna (1984)[FL1984] addressed the single-depot single-vehicle routing in SRP. The developed algorithm is based on graph theory and finding an Eulerian circuit which is subjected to street priorities ensuring that the main streets are ploughed first and other streets in decreasing order of their importance. The model is described for undirected graph representation of the road network but can be adapted to the directed case as well.

A multi-objective heuristic for routing problem of snowploughs in rural environment is presented by Haslam and Wright(1991)[ER1991]. The problem was defined as generating a set of routes with minimal cost which is measured by the number of used snow-ploughs and the total distance traveled. The underlying road network contains road segments that do not belong to the operational district.
under consideration and they do not need to be treated. An ad hoc algorithm developed by them is a multi-step procedure in which a route originating from, some seed node is grown successively by adding feasible edges to the route having a predefined class and length. The seed nodes are selected by experts having the knowledge of the geographical configuration of the network. Once the route is initiated with non-treated edges whose selection do not force the route over its maximum distance constraint and has the same class as the route is added to the route. If the route is not ended to the depot, the shortest path is added back to the depot by deadheading. Intelligent choices of seed nodes and by varying number of vehicles, the user should be able to reach a solution in polynomial time.

Campbell and Langevin (1995b) [FA1995b] formulated the snow removal problem as a multi-resource generalized problem where the objective is to minimize the total transport cost from sectors to disposal sites, weighted by the annual volume of snow, subjected to capacity constraints on disposal cities and assigning each sector to exactly one site. Due to the inherent complexity (NP-hard) of the problem, they suggested a two-phase heuristic. The first phase assigns each sector to a disposal site by a penalty-based assignment procedure and in the second phase two-opt exchange procedure is carried out to assign the sectors and sites pairwise to decrease the objective value.

While studying the routing problems we come back to one of the most well-known problem - The Shortest Path Problem, SPP, i.e. the problem of finding the shortest path between pair of nodes in the network. The most efficient algorithms for solving the SPP were presented by Dijkstra and Ford. The Dijkstra’s algorithm is applicable to graphs with non-negative edge weights and that weights could be edge transportation time or transportation cost or edge length. In our solution we have set the weights to the transportation time of the edge.

A well-known node-oriented problem and its various versions, related to our problem is the Vehicle Routing Problem, which is explained in the next section.

2.3 Vehicle Routing Problems

Vehicle Routing Problem has a central place in distribution management. It is also an important combinatorial optimization problem that came up with several powerful exact and approximated solution methodologies. The VRP is also an NP-hard [Jonson1979] problem that is extremely difficult to solve to optimality. No exact algorithm so far can consistently solve the VRP instances in excess of 50 cities, although several larger instances, some involving more than 100 cities, have been solved optimally [Gil]. So, the only practical approach is the use of heuristics. These belong to two broad characteristics: classical heuristics and meta heuristics. Classical heuristics again is classified into three categories: Constructive heuristics, two-phase heuristics, and improvement methods. Six main types of meta heuristics have been applied to the VRP. They are Simulated Annealing, Deterministic Annealing, Tabu Search, Genetic Algorithms, Ant Colony Optimization, and Neural Networks. From all these categories we shall be using the Simulated Annealing to come up with the optimized solution to route the available snow ploughing vehicles within the network. The reasons that motivate us to use SA as our optimization algorithm will be elaborated more along with the algorithm itself later while discussing mathematical formulation. A general SA procedure starts with an initial solution \( x_t \) and move at each iteration \( t \) from \( x_t \) to a solution \( x_{t+1} \) in the neighborhood \( N(x_t) \) of \( x_t \), until a stopping condition is satisfied. If \( f(x) \) denotes the cost of \( x \), then \( f(x_{t+1}) \) is not necessarily less than \( f(x_t) \). As a result, care must be taken to avoid cycling. The purpose of that brief description is to bridge the connectivity between between the snow-ploughing routing problems to the vehicle routing problems through the classical and modern heuristics approach.

In this thesis we focus on solution methods of meta heuristics types, particularly SA. We believe and most of the work on large scale problems indicate that committing on this type of solution

---

[3] NP-hard (Non-deterministic polynomial-time hard) is a class of problem where a known polynomial-time algorithm is there for non-deterministic machine to get an answer.

[4] A method that is used rapidly come to a solution that is hoped to be close to the best possible answer.

[5] Kind of heuristic used in combinatorial optimization problem and they try to escape the local optimum to reach the global optimum.
method is a good decision. The problem is to find a set of routes with associated time period for a number of available snow ploughing vehicles that minimize the total cost of the snow removal operation. Once the route is generated, it may be operated for a period of several years with some minor modification if new path is introduced. Therefore it is interesting to find a solution to route of snowploughs in a more systematic manner. In doing so we are interested to develop an application framework for snow-ploughing problem that was already investigated based on different models.

2.4 Various types of Snow Removal Routing Problem

In practice, there are two types of snow removal problem based on time restrictions in form of time window within which road segments need to be ploughed. That gives rise to more restricted version SRRPTWAS referred to as Snow Removal Routing Problem With Time Windows After Snowfall. Another version of the SRP is the Snow Removal Routing Problem With Time Windows During Snowfall, SRRPTWDS, according to which the snowploughs starts the removal of the snow during snowing conditions. Assuming that there is no time horizon (the length of time the snow fall will continue) on the snowfall, the standard classes of the road segments then turn into frequencies by which the road segments need to be ploughed. Even though both of the cases are of interest from the operational and mathematical point of view, we shall be considering the latter type (SRRPTWDS) during this thesis work.

Periodic SRRPTWDS is also NP-hard as VRP[Golam2004], and as a result we focus our interest on meta-heuristic solution methods, i.e. methods that hopefully give good solutions, but are not guaranteed to give an optimal solution. Furthermore, if such methods happens to find an optimal solution, it cannot be verified that the solution is optimal.

2.5 The Networks

The developed solutions in this thesis are implemented on the Sioux Falls and Eskilstuna networks. The first one corresponds to the simplified version of the Sioux Falls, SF, road network located in South Dakota, USA and the latter one is the Operational District of Eskilstuna, ODE. We used SF to test all the algorithms we have generated, because of its lesser node and edge complexity.

2.5.1 The Road Network of Eskilstuna

The national road network in Sweden is divided into several operational districts. That includes ODE, which is located in the middle of Sweden and is composed of 1454 road segments and 353 road junctions[Golam2004]. A road segment of the network has a certain set of attributes such as length, end points, direction, priority class (that is translated into the time window for ploughing of the road segment), average annual daily traffic. The end points are types of road junctions, roundabouts or where the roads actually ends.

In the mathematical representation of the Network, a road segment corresponds to an edge with an associated set of attributes; a node represents various types of traffic junctions[Golam2004]. The

\[\text{amount of allowed time to plough an edge between a pair of nodes}\]
geographical map of ODE is depicted in figure 2.3 [Golam2004]:

![Geographical map of Eskilstuna](image)

**Figure 2.3: Operation district of Eskilstuna**

The size of the underlying mathematical network affects the choice of solution methods for routing problems, therefore a pre-processing phase is introduced in which the original network is reduced, if possible, with respect to both traffic junctions and road segments. The reduction is based on joining or discarding of edges and removing of nodes. The construction of the mathematical representation has been a dynamic process where translation of junctions to the nodes and removing or inserting road segments from the road network have been performed while developing the solution methods. The result is two scenarios with different conditions and they are:

- After snowfall.
- During snowfall.

As specified before, we are only concerned about the scenario *During snowfall* in this thesis work.
2.5. THE NETWORKS

2.5.2 During Snowfall

While constructing the mathematical representation of the network for the case during the snowfall, we have chosen to represent the two-way directed edge as one-way edge path segment as shown in figure 2.4a and figure 2.4b:

![Diagram showing one-way and two-way directed edges.](image)

**Figure 2.4**: Edge representation in mathematical model

The figure 2.4 shows how we represent both type of edges with one-way direction. And that formulation helped us to design our problem easily which will be elaborated more in the implementation chapter. The figure 2.4a represents a one-directed edge path which we call one one-way directed path segment. Figure 2.4b shows a two-way directed edge which we call one two-way-directed path segment. There are road segments that are included to make the network connected and do not need to be ploughed but can be used for transportation.

2.5.3 Applied Optimization and Operations Research

A mathematical model in applied optimization is an idealized representation expressed in terms of mathematical symbols and expressions [Golam2004]. In general a mathematical model is constructed as follows:

Optimize $f(x_1, x_2, \ldots, x_n)$

subject to

$h_1(x_1, x_2, \ldots, x_n) \leq b_1$
$h_2(x_1, x_2, \ldots, x_n) \leq b_2$

..............................

$h_m(x_1, x_2, \ldots, x_n) \leq b_m$
$g_1(x_1, x_2, \ldots, x_n) \geq c_1$

..............................

$g_k(x_1, x_2, \ldots, x_n) \leq c_k$
$y_1(x_1, x_2, \ldots, x_n) = d_1$

..............................

$y_p(x_1, x_2, \ldots, x_n) = d_p$

$x_1 \in X_1, \ldots, x_n \in X_n$

The term optimize means minimization or maximization of a function $f(x_1, \ldots, x_k)$, called the objective function, which is an appropriate measure of performance. The set of decisions are represented as decision variables, $x_1, x_2, \ldots, x_n$. Every decision variable stands for a quantifiable decision to be made and whose respective value has to be determined. Any restriction on the value that can be assigned to the decision variables are expressed mathematically in form of constraints i.e.
CHAPTER 2. THEORETICAL BACKGROUND

The objective function and the constraints are functions of decision variables. All the predefined values and constants are parameters. Finally, $X_1, \ldots, X_n$ specify the set of values for each variable. The advantage of mathematical model in comparison to a verbal description of the problem is that the mathematical model describes the problem in a concise manner. It makes the overall structure of the problem be more comprehensible, and helps to reveal important cause-and-effect relationship. It also facilitates dealing with problem in its entirety while considering all interrelationships. Last but not the least, a mathematical model provides with a link between the use of mathematical techniques and advanced computer science, that helps to analyze the problem efficiently.

There is no special method, that solve all optimization problems. The techniques, algorithms, and methods, together with the mathematical theories that they are based on, used to identify an optimal solution to the mathematical models, constitute the subject of mathematical programming. Mathematical programming is a part of wider branch of science, *Operation Research*, which includes wide areas of application of scientific methods and techniques to complex decision problems\cite{Golam2004}.

According to Nima\cite[pp. 12-14]{Nima2001}, *Operation Research* targets to determine the best policy, strategy, or decision from several options. Route planning, distribution, traffic design, marketing, finance, transport, telecommunication, management and scheduling are some of the important application areas of *Operations Research*. During the last decades, there has been an impressive progress in the field of *Operations Research* due to the parallel development of modern computers.

Nima\cite[pp. 12-14]{Nima2001} states some steps that are included in a typical *Operations Research* study.

- **Formulate the Problem**
  - Study the relevant system.
  - Develop a well-defined statement of the problem.
  - Recognize the appropriate objectives and restrictions on what can be done.

- **Construct the Mathematical Model**
  - Reformulate the outcome of the previous step into a form that is convenient for analysis.
  - Construct a mathematical model that represents the essence of the problem.

- **Derive a Solution to the Model**
  - Utilize the standard mathematical solution methods and algorithms.
  - Implement the solution methods in high-level programming languages, that becomes a standard part of this phase in most operation research studies. A common theme in *Operations Research* is the search for an optimal, or best solution. Needless to say that solution only with respect to the model being used. There is no guarantee that the best solution in the model is also the best solution in real-life. There are often too many uncertainties associated with the real-world problem and the model covers all these factors.

- **Test and Evaluate the Model and the Solution**
  - Evaluate the model and the obtained solution to the model.
  - Re-examine the formulation of the problem, check and test for errors or oversights in the model and make sure that all mathematical expressions are correctly used and expressed.
  - Verify the obtained solution and investigate the effect by changing some of the factors.

- **Implementation of the Solution.**
  - Implement the final solution to the real-world problem.
  - This level is executed when the level of accuracy of the solution is accepted to improve the result further.
In addition, it is very rare to identify each of these phases as distinct and separate. They usually influence one another and may be partially overlapping in duration. The above steps aligns nicely with the software engineering aspect of development. If we need to map the above steps in a software engineering perspective we can derive the following relationship:

- Problem formulation & mathematical model construction corresponds to Requirement analysis and design.
- Derivation of solution model relates to system design and implementation.
- Testing and evaluation of the model directs to different hierarchies of testing in software.
Chapter 3

Mathematical Formulation

This chapter will provide a mathematical description and propositional phase of the concepts introduced in Chapter 2. We shall start by describing the mathematical approaches that have been followed to generate the optimized solution of the problem. We study the periodic formulation of the SRRPTWDS problem. Finally we study the core algorithm of the problem - Simulated Annealing and motivate from different perspectives to choose this one.

3.1 Overview and solution approaches

Even though the focus of this thesis work is The Snow Removal Routing Problem with Time Window During Snowfall, we need to have a little overview of The Snow Removal Routing Problem with Time Window After Snowfall for the clarity of the problem. The Snow Removal Routing Problem After Snowfall, SRRPAS, is the mathematical problem of routing a set of snowploughs after snowfall, so that every road segment is ploughed exactly once and the total cost of routing is minimized. This means that a route in the solution is defined either with vehicles available or with more than available vehicles and penalty cost. The penalty cost is associated because the solution uses more vehicles than it has been allocated with. A route starts at a depot followed by a sequence of road segments and returns to the origin. Each road segment in the route is either ploughed or simply used for transportation.

In real-life there are additional restrictions on the solution that has to be considered. According to Chapter 2, every road segment belongs to a standard class defined for each road segment. That implies that a time window on each road segment must be satisfied, which gives rise to a more restricted version of SRRPAS referred to as the snow removal routing problem with time windows after snowfall, SRRPTWAS.

So SRRPTWAS is the problem of designing a set of routes for snowploughs at a minimal cost that together cover all the road segments which are to be ploughed within specified time limits. The solution is given by the routes and the time schedule, i.e. the start and finishing time of ploughing each road segment. The design of these routes is one of the factors affecting the efficiency and cost of the snow removal operation.

Let $G = (N, A)$ be a directed network with node set $N$ and edge set $A$. Each edge $e = (i, j) \in A$, has an associated set of data indicating the ploughing cost, ploughing time, transportation cost and transportation time of the processing of the road segment. We define a directed route $r$ in $G$ as a sequence of $(e_1, e_2, \ldots, e_n)$ of edges, where the end node of edge $e_{k-1}$ is the start node of edge $e_k, k = 2, \ldots, n$. In addition, a route indicates whether the edges in the path are ploughed or just used for transportation and the corresponding amount of time( aggregation of ploughing and transportation time ) to cover that edge segment.

Let $D$ be the set of depots in the network, $R$ the set of all feasible routes with respect to time windows on the road segments, originating at depot $d \in D$, and $A_p \subseteq A$ the set of road segments that need to be ploughed.
Let $v_d$ is the number of vehicles available at depot $d$, $w_d$ is the charge for using one extra snow plough in addition to those available at depot $d$, and $c_r$ the operating cost of route $r \in R$. Further let,

$$a_{er} = \begin{cases} 
1 & \text{if edge } e \in A_p \text{is ploughed in route } r \in R \\
0 & \text{otherwise} 
\end{cases}$$

$$x_r = \begin{cases} 
1 & \text{if route } r \in R \text{is a member of minimal cost route set} \\
0 & \text{otherwise} 
\end{cases}$$

According to Gholamreza (2004, pp. 20), the problem SRRPTWAS can be formulated as follows:

$$\min_{x,y} \sum_{r \in R} c_r x_r + \sum_{d \in D} w_d y_d$$

(3.1)

$$\sum_{r \in R} a_{er} x_r = 1 \quad \forall e \in A_p,$$

(3.2)

$$\sum_{r \in R_d} x_r \leq v_d + y_d \quad \forall d \in D,$$

(3.3)

$$x_r \in \{0,1\} \quad \forall r \in R,$$

(3.4)

$$y_d \geq 0 \quad \forall d \in D,$$

(3.5)

Due to the constraint in (3.3), the problem (3.1) can be regarded as a Constrained Set Partitioning Problem, CSPP. Each column $a_r$ of the constraint matrix $A = (a_r)_{r \in R}$ can be viewed as a subset, where $e \in a_r$ if and only if $a_{er} = 1$ and the constraints state that every edge has to be covered only once. There are no explicit constraints imposed on satisfying the time criteria on the edges. Instead, it is assumed that the columns in $A$ are feasible with regard to satisfying the time constraints.

In general most snowfall conditions require snow-ploughing of roads before the end of the snowfall to make the roads traffic-able, gives rise to the Snow Removal Routing Problem with Time Window During Snowfall, SRRPTWDS. In contrast to the SRRPTWAS, the road segments needs to be ploughed repeatedly during the snowfall while respecting the standard time window associated with each edge segment. Two versions of SRRPTWDS are considered of which one is when a time horizon (the length of time that the snowfall will continue) for the duration of the snowfall is known and the other, the snowfall is thought to be continuing for an unpredictable amount of time. The first version is considered as a routing problem with dynamic time windows, while the other consider the time windows to be transformed into frequencies by which the road segments need to be ploughed. The time horizon version is somewhat similar to SRRPTWAS, while the periodic case differs significantly. Solving the periodic case of SRRPTWDS is more complex and time consuming compared to SRRPTWAS and solving the problem requires the design and consideration of new algorithms.

The snow-plough fleet consists of several types of vehicles, distinguished by their capacity in ploughing snow or/and capability of spreading salt while ploughing. We are concerned with two types of vehicles of which one can be used for salting while ploughing and the other is only for ploughing. A vehicle with salt equipment is permitted to plough every road segment in the network.
3.2 Time Periodic Formulation of SRRPTWDS Problem

This section presents a version of the problem where, we are not aware of any time horizon. In this case, we are interested of feasible routes that can be repeated during snowfall conditions. The following descriptions are based on the ideas extracted from Gholamreza(2005, pp.64).

Let $G = (N, A)$ be a directed network with the node set $N$ and edge set $A$, where $A = A_p \cup A_T$. The edges $A_p$ are those, which must be ploughed and the edges in $A_T$ are auxiliary edges corresponding to the road segments that may or may not be the part of the operational district under consideration, but added to construct a complete network and to facilitate transport.

Let further, $G^k = (N^k, A^k) \subseteq (N, A)$ be a directed network with node set $N^k \subseteq N$ and edge set $A^k \subseteq A$. The edge set $A^k$ includes edges, that must be ploughed by a vehicle of type $k \in K$. These edges are denoted by $A^k_p$. The remaining edges, $A^k - A^k_p$, are transport edges, which are those edges in $A_T$ that may not be ploughed by a vehicle of type $k$ and those in $A_T$.

Let $G$ be the directed network. We define directed route $r$ in $G$ as a sequence $(e_1, e_2, \cdots, e_{n_r})$ of $n_r$ edges where, the end node of edge $e_{k-1}$ is the start node of edge $e_k$, $k = 2, \ldots, n_r$. A route also contains an indication if the edges in the path are ploughed or just used for transport while traveling through the shortest path from the end node to the start node and the amount of time taken to cover (plough/transport) that edge. For an edge $e$ that must be covered every $m$ hours. Further, let a route of type $k \in K$ be a route in which all the road segments must be operated by a vehicle of type $k$. A route, must be associated with one and only type of vehicle.

Each vehicle with a certain type starts operating from some node, ploughs all the road segments that require ploughing within the time period and returns back to the starting node. We call that a round - the most important part of the solution. Each vehicle repeats its round during snowfall to maintain the road segments trafficable with respect to the standard service level conditions set by SNRA.

In our model we have reflected the real-life network. Usually, there are two traffic lanes, i.e. they are two-way road segments. Hence, in the new network for each pair of nodes in $i$ and $j$, presence of a forward edge $(i,j)$ implies that there is a back ward edge $(j,i)$. This might require some modifications of the underlying network. At the same time the network may contain mixture of single way and two-way directed road segments which we have to address as well.

While following the model, we need to introduce new notations and definitions as follows:

Definition 1 A directed walk in the directed network $G_d = (N, A)$ is a sequence $(n_0, e_0, n_1, \ldots, e_{k-1}, n_{k-1})$ whose terms are alternately nodes and edges such that, for $0 \leq i \leq k-1$, the edge $e_i$ has a start-node $n_{i-1}$ and and end-node $n_i$.

Definition 2 A ploughing round in $G_d$ is a closed directed walk plus an indication for each edge in the walk, if the edge is ploughed or just used for transport. A proper ploughing round in $G_d$ is a periodic ploughing round in which edge $(i,j)$ is either ploughed or just used for transport.

3.3 Simulated Annealing

The concepts of annealing in combinatorial optimization was introduced by Kirkpatrick, Gelatt & Vecchi in the early 1980. These concepts are based on a strong analogy between the physical annealing process of solids and the problem of large combinatorial optimization problem. In this section we follow this analogy in order to introduce the simulated annealing algorithm from an intuitive point of view.

Annealing is known as a thermal process for obtaining low energy states of a solid in a heat bath[Boltzman1989]. And the process contains the following 2 steps:

- Increase the heat bath to maximum value at which the solid melts.
- Decrease carefully the temperature of the heat bath until particles arrange themselves in the ground state of the solid.
All particles in the liquid state arrange themselves in a random manner. In the ground state the particles are arranged in a highly structured lattice and the energy of the system is minimal. The ground state of the solid is reached only if the maximum temperature is sufficiently high and the cooling is done slowly. Otherwise the solid will be frozen into a meta-stable state rather than into ground state. This physical annealing process can be modelled successfully by using computer simulation methods from condensed matter physics.

Certain optimization problems become unmanageable using combinatorial methods as the number objects becomes large. Therefore we assume an analogy between a physical many-particle system and combinatorial optimization problem with following equivalences [Boltzman1989]:

- Solutions to a combinatorial optimization problem are equivalent to states of a physical system.
- The cost of a solution is equivalent to the energy of the state.

For example, in Snow Removal Routing Problem , a snow plough must visit some large number of junctions/nodes while minimizing the total mileage traveled. If the snow plough start at a random point, can plough as many path segments as possible until the time constraint associated with the period is not breached, hoping to reduce the number of periodic routes. The difficulty with this approach is that it rapidly finds a local minimum, it cannot get from there to global minimum.

Simulated Annealing improves the strategy through the introduction of two tricks. The first is called the so-called Metropolis Algorithm, in which some solution that do not lower the cost are accepted when they serve to allow the solver to explore more of the possible space of solutions. Such bad costs are allowed using the criterion that

\[ e^{-\Delta D/T} > R(0,1) \]

where \( \Delta D \) is the change in solution cost, (negative for good solution, and positive for bad solution), \( T \) is the synthetic temperature, and \( R(0,1) \) is the random number in the interval \([0,1] \). \( D \) is called the cost function and corresponds to the free energy in the case of annealing a metal (the temperature parameter would actually be the \( kT \), where \( k \) is the Boltzman’s constant and \( T \) is the physical temperature, in the Kelvin absolute temperature scale). If \( T \) is large, many bad solutions are accepted, and a large part of the solution space is accessed.

The second trick again by the analogy with annealing metal, to lower the temperature. After making many periodic routes and observing the cost function declines only slowly, we lower the temperature in a controlled manner, and thus limits the size of allowed bad solution. After lowering the temperature several times to a low value, one may then quench the process by accepting only good solution in order to find the global minimum of the cost function.

A great variety of problem-specific or more general solutions have been developed in the recent decades. Here we focus on meta-heuristic techniques, and in particular one of them, namely, simulated annealing is a technique which seeks good (i.e. near-optimal) solutions at a reasonable computational cost without being able to guarantee either feasibility or optimality, or even in many cases to state how close to optimality a particular solution is.

It has been proved that if we carefully control the rate of cooling of the temperature, SA can find the global optimum. However, this requires infinite time. Fast annealing and very fast simulated re-annealing(VF SR) or adaptive simulated annealing(ASA) are each in turn exponentially faster and overcome that problem.

### 3.3.1 The method

The SA algorithm uses a random search which, not only accepts changes that decrease the objective function \( f(x) \), but some changes that increase it. The latter are accepted with the probability

\[ p = \exp(-\delta f/T) \]
where $\delta f$ is the increase in $f$ and $T$ is the control parameter. The implementation with the basic SA algorithm is straightforward. The figure 3.1 shows the structure:

![Flowchart of Simulated Annealing algorithm](image)

Figure 3.1: The structure of Simulated Annealing algorithm

To initiate the above algorithm we have to provide the following elements:

- A representation of possible solutions which we shall present in the next chapter.
- A generator of random changes in solutions.
  - The changes will be generated by some functions that remove, insert, split and merge the plough path segments in the periodic route.
- A means to evaluating the problem functions and
  - We have attributes associated with each edge path segment and vehicle which, will assists to quantify the periodic route.
- An annealing schedule - an initial temperature and rules for lowering it as the search progresses.

As we have specified before that periodic SRRPTWDS is NP-hard and there are other methods to use to come up with a solution for NP-hard problems. Here we try to motivate our reason of choosing SA as our solution process. The reasons are:
• SA can deal with highly non-linear models.
• Its main advantages over other local search methods are its flexibility and its ability to approach global optimality.
• The algorithm is quite versatile since it does not rely on any restrictive properties of the model.
• SA methods are easily tuned.
• SA can deal with arbitrary systems and cost functions.
• The empirical data shows that SA can deliver optimal or near to optimal solution.
• SA is relatively easy to code, even for complex problems.

Along with the above strengths there are some weaknesses of SA as well. They are:
• There is a clear trade-off between the quality of the solutions and the time required to compute them.
• The precision of the numbers used in implementation is of SA can have significant effect upon the quality of the outcome.

3.3.2 Comparison with other methods

Any efficient optimization algorithm must use two techniques to find a global maximum. They are:
• Explore to investigate new and unknown areas in the search space.
• Exploit to make use of knowledge found at points previously visited to help find better points.

These two requirements are contradictory, and a good search algorithm must find a trade-off between two. The tables 3.2 and 3.3 will find the comparison between several other meta heuristic approaches with SA[Ana].

<table>
<thead>
<tr>
<th>Neural sets</th>
<th>SA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Neural sets learn how to approximate a function</td>
<td>SA searches for global optimum</td>
</tr>
<tr>
<td>It is flexible function approximators</td>
<td>It is an intelligent random search method</td>
</tr>
<tr>
<td>Adaptive characteristics of neural sets are a huge advantage in modeling changing environments</td>
<td>The power-hungriiness of SA limits its use as a real-time application</td>
</tr>
</tbody>
</table>

Table 3.2: Comparison table between Neural sets and SA

<table>
<thead>
<tr>
<th>Genetic algorithm</th>
<th>SA</th>
</tr>
</thead>
<tbody>
<tr>
<td>GA is not developed as a optimization algorithm</td>
<td>SA is developed for optimization</td>
</tr>
<tr>
<td>GA does not offer any statistical guarantee of global convergence to an optimal point</td>
<td>SA provides some statistical guarantee that the solution may not better but will not be worse than the initial solution.</td>
</tr>
</tbody>
</table>

Table 3.3: Comparison table between Genetic algorithm and SA

In general, SA implementation is a simple and fast algorithm that solves many VRP problems to near optima and periodic SRRPTWDS is also kind of VRPs. Due to the global best approach in local neighborhood search, the algorithm is able to result in stable local optimal solutions almost at all the times.
Chapter 4

Initial Solution Generation

This chapter will provide with algorithmic analysis of the initial solution generation upon which simulated annealing will start working to come up with better solution. Two different types of algorithms has been analyzed and used to generate that solution. One type deals with only two-way directed edges in the network and the rest deals with only single-way directed edges in the network. We discuss both of them and it is followed by introducing some other important algorithms used by SA in a repeated manner to reduce the number of periodic routes generated by initial solution.

4.1 Problem Formulation

The periodic SRRPTWDS is given by a set of vehicles $V$ categorized by each of their capabilities, a special node depot, where the vehicles start off from, a set of nodes $N$ to be visited, a directed network connecting the depot and the nodes in the network. Let us assume that there are $K$ vehicles, $V = \{0, 1, 2, \ldots, K-1\}$ and $N$ nodes. Each edge in the network corresponds to a connection between two nodes. A period route is defined as starting from a depot, going through number of nodes in the network and ending at the depot. The number of periodic routes in the traffic network is equal to the number of vehicles that is used. A cost $c_{ij}$ and travel time $t_{ij}$ are associated with each edge of the network traveled by the vehicle. That going through could be done either by ploughing the edge path segment or by traveling through shortest path.

Let’s try to understand a scenario shown in figure 4.1:

![Figure 4.1: Periodic route formation procedure](image)

The red-colored edge in figure 4.1a is the very first edge that we want to plough and has a time-window of 2 hours. We manage to plough it within 30 minutes and we set the period of the route to 2 hours. So the initial path becomes $1-P-2$ and the accumulated ploughing time is 30 minutes.
CHAPTER 4. INITIAL SOLUTION GENERATION

From now on we have to make sure that we come back to the start node (1) of the red-colored edge by one of the following ways within 2 hours:

- Plough.
- Shortest path transpiration.
- Mixture of the plough and transport.

Each of the path between two nodes has time window of 2 hours. From node 2 we have outgoing edge which takes 10 minutes to plough and it has a time-window of 2 hours. We add that ploughing time to our accumulated ploughing time which becomes now 40 min. And we calculate the shortest path transportation time from node 6 to start node 1. If the accumulated plough time and shortest path time is less than the period, then we add the path between 2 and 6 to the periodic route and the path becomes 1-P-2-P-6. We look for outgoing edges from node 6 and consider the edge between node 6 and 7. We follow the same procedure as we have done to include the path between node 2 and node 6 in the periodic route. If the conditions are satisfied the path becomes 1-P-2-P-6-P-7. In figure 4.1a we manage to cover all the path segments within time constraints and the ultimate path became 1-P-2-P-6-P-7-P-1.

In figure 4.1b we have another scenario where we do not manage to plough all the path segments because of the time constraints. We start with the path segment between 1 and 2 and the initialize the periodic route with 2, since this is the very first path segment we encounter. And we follow the procedure as we have described before to include new path into the periodic route. We manage to plough from node 1 to node 7, which means that the accumulated ploughing time from node 1 to node 7 and the shortest path transportation time from node 7 to node 1 is less than the period hours (2). But when we try to plough the path between 7 and 1, we total accumulated ploughing time exceeds the period of the route. So the path becomes 1-P-2-P-6-P-7-T-1.

4.2 An Initial Solution

Most meta heuristics involves finding an initial feasible solution and then improving on that solution using local or global optimization techniques. Here we have made two different types of algorithms for two different type of edges to generate the initial solution. The first type of the initial solution deal only two-way directed edges in the network and the second type deal with only single way directed edges in the network. Let us assume that a periodic route \( R_p = \{N_1, N_2, \ldots, N_m\} \) where \( N_1 \) is the first node and \( N_m \) is the last node. The feasibility of inserting a node in the periodic route \( R_p \) is checked by inserting the node between all the edges in the current route and selecting the edge that has the lowest time window. Now we formulate the algorithm that we have followed to generate an initial solution for two-way directed edges in the network. Initially all the nodes, edges and their corresponding attributes are loaded from the database in the main memory.

The algorithm 4.1 goes through all the edges in the memory and treat only two-way directed edges in the network. We treat only those edges which do not have the following characteristics:

- The edge path segment is already ploughed.
- The edge path segment has breached the period of the route and it can not be included in the current periodic route.
- The edge path segment can only be used as transport path.

Most of the two-way directed edges in the network are both meant to be ploughed and salted and there are few that are meant to be ploughed only. And based on the requirements of the edges we assign a vehicle with the right competence. If a two-way directed path segment is meant to be both ploughed and salted then we have to get a vehicle that can plough and salt, whereas, we assign a
Algorithm 4.1 Initial Solution Generation for two-way directed edge

While (the end of the edge list is not reached)
{
    if (the edge is both way)
    {
        if (the edge is not ploughed yet AND
            the edge cannot be ploughed AND
            the edge is not a TRANSPORT edge)
        {
            if (the edge is meant to be both ploughed and salted)
            {
                Get the available vehicle that can both plough and salt.
            }
            else if (the edge is meant to be ploughed only)
            {
                Get the available vehicle that can plough only.
            }
        }
        Get the edge ploughing time covered by the vehicle.
        if (edge ploughing time is less than the period of the edge)
        {
            Create a new Periodic Route starting with the edge and reversely
directed edge.
            Set both the edge status to PLoughED.
            Check through the edge list in a recursive manner to look for
            neighboring two-directed edges that are ploughable based on
time and competency constraint.
        }
    }
}
Create a Solution and assign the periodic route path list to the Solution’s
path list.
Return the Solution.
plough capable vehicle to a two-directed path segment that is meant to be ploughed only. It is quite fine to assign a vehicle that is capable of both ploughing and salting to a two-way directed path segment that needs to be ploughed only, but we try to avoid this situation, because it will increase our cost. Then we browse through the edge list in depth-first search manner and add edge path segment to the periodic route until the period is breached. The figure 4.2 will give a brief overview of the traversal operation:

![Figure 4.2: initial solution generation](image)

The edge-pair in red color is the first we encounter in the edge list after reading from the database and if that edge pair is meant to be ploughed or both ploughed and salted, we assign the appropriate vehicle for that edge pair. Next during the traversal we consider only those edge pairs that match the vehicle competences. Then we start traversing from one of the end nodes and add edge pair to the periodic route until we do not find anymore two-way directed edge path segment left to be ploughed or the time period constraint is already reached. In the above figure we continue to the right side after treating edge $e_1$ and $e_2$ and we keep including edge $e_3$ and $e_4$ (blue color) to the periodic route until we reach the end node for edges $e_9$ and $e_{10}$. From that end nodes we have no more two-way directed neighboring edges to treat, so we back-track and look for rest of the other two-way directed edges. After that repeated back-tracking we come back to the initial position where we started from and if we still have time left, we start treating the other end of the node at the initial edge-pair we have started with and do traversal as we have just described.

Now we shall provide an overview of another algorithm that we use to generate initial solution with only one directed edges. The data-flow diagram in figure 4.3 will be followed by a brief description of the steps we have gone through to create a periodic route. The figure 4.4 expands the green activity box of the figure 4.3:

The activity diagram in figure 4.4 shows how we initiate the traversal operation for only one-way directed edges in the network. While choosing the neighboring edge of the current edge we check for the similarity in the time-window in the neighboring edges and choose the one with the nearest similar as in figure 4.5:
4.2. AN INITIAL SOLUTION

Figure 4.3: Generation of Initial Solution (initial version)

Figure 4.4: Expanded activity diagram of Initial Solution Generation
CHAPTER 4. INITIAL SOLUTION GENERATION

Figure 4.5: Neighboring edge choosing criteria

The red colored edge in figure 4.5 is the current one-way directed edge that we are currently considering and we are trying to figure out which neighboring edge should we consider. The green-colored nodes are the most ideal candidate to be included in the periodic route than the rest of the other nodes, since they are all similar to the time window of the current edge (red color). We have to choose one edge from these three one-way directed edges that are connected to the green-colored nodes based on the least shortest-path time taken from all the three nodes back to the starting node (red-colored) of the current edge. We calculate the shortest path using Dijkstra's shortest path algorithm. Figure 4.6 explains the procedure:

(a) Calculation of shortest path from each green node back to the red node where all the dotted lines arrive
(b) Green node that has the red colored edge arriving towards it turned out to be the neighboring edge

Figure 4.6: Neighboring edge choosing criteria based on shortest path

As we can see from figure 4.6a that we calculate the shortest-path from all the green nodes back to the starting node of the red colored edge and we take the one with the least-shortest path time. The resulted path is shown in figure 4.6b. And we keep on doing like that until time period of the periodic route is not violated. When there is no time left, the periodic route contains a list of path segments with ploughed and transport path.

After the successful generation of the initial solution we come up with a list of several periodic routes where each edge is ploughed by each vehicle of respective type based on competences. At that position the simulated annealing has a solution to get started with and we call it the initial solution.
4.3 Periodic Route Path Segment Removal:

Simulated annealing will be using several other utility functions to operate on the initial solution and try to optimize the solution by reducing the number of periodic routes. Periodic route path segment removal is one of them. It could be several path segments or just a single path segment. In this project we remove only single path segment and that segment could be one-way directed or two-way directed. We remove only those path segments that are already ploughed and so when we remove any plough path segment from the periodic route we just replace the status of that path segment to transport which signifies that the path has been removed. The figure 4.7 shows the process:

![Diagram](image)

Figure 4.7: Path segment removal from Periodic Route

The figure 4.7a is a one-way directed edge and the plough path segment has been identified with red color which is going to be removed, the figure 4.7a shows how the very same edge status has been changed to transport. The same procedure is applicable to two-way directed edge as shown in 4.7b.

4.4 Periodic Route Path Segment Insertion:

The ploughed path segment that is removed as discussed in the previous section has to be inserted into path segments of another periodic route. Now the insertion process would be different based on the direction of the path segments. So we shall call the path segments that is going to be inserted as the guest path segments and the periodic route that will be containing the new path segments along with the existing ones after successful insertion as host path segments. Host path segments could be one-way directed or two-way directed or mixed type which means that the host periodic route contains both one-way and two-way directed path segments. Guest path segments can be only one-way or two-way directed. So we can categorize the insertion in the following manner:

- Two way directed host path segments and two way directed guest path segment.
- Two way directed host path segments and one way directed guest path segment.
- One way directed host path segments and two way directed guest path segment.
- One way directed host path segments and one way directed guest path segment.
- Mixed way directed host path segments and one way directed guest path segment.
• Mixed way directed host path segments and two way directed guest path segment.

In all of the above cases we have to calculate the shortest path and time from each and every node in the host path segments to the start node of guest path segment and take the one that has the least shortest path time. The host path segments may consists of only plough paths or mixed of plough and transport paths. We shall never encounter any periodic route with only transportation paths, because periodic routes with only transportation paths are of no interest and we remove those periodic routes before any further processing within the simulation. And the guest path segments will always be plough path. All the figures that follows will use some notations and their elaborations are:

• P - plough path.
• T - transport path.
• SPn - nth shortest path.

![Diagram](image)

(a) Calculation of shortest path from each node of the host path segments to the start node of a guest path segment
(b) After successful insertion process the blue path segment shows the connectivity of newly inserted path segment

Figure 4.8: Insertion of a two-way directed guest path segment into two-way directed host path segments of a periodic route

The figure 4.8a above shows how we are calculating the shortest path (green dotted line) from each node in the host path segments to the start node of the guest path segment and we are considering the path with the least shortest path time. The figure 4.8b shows that we have found the shortest path (blue line) based on least shortest path time from one of the host nodes (green color) to the start node (green color) of guest path segment. Then we calculate the shortest path back to node where we have found the shortest path from in the host path segments. While doing the insertion we update the period of the route and the whole insertion process is only possible if the total time (aggregation of plough time and transport time) is less than the period of the periodic route of the host path segments.

There is something extra we have to do in figure 4.9 when the host path segments are two-way directed and the guest is one-way directed path segment. Along with the shortest path calculation from the start node in the guest path segment to the node (green color) in the host path segments, we have to calculate the shortest path from the end node in the guest path segment to the node (green) in host path segments as shown in figure 4.9b.
4.4. PERIODIC ROUTE PATH SEGMENT INSERTION:

(a) Calculation of shortest path from each node of host path segment to start node of one-way directed guest path segment.

(b) After successful insertion process, blue path segment shows the connectivity of newly inserted path segment.

Figure 4.9: Insertion of one-way directed guest path segment into two-way directed host path segments of periodic route.

---

(a) Calculation of shortest path from each node of the host path segment to the start node of two-way directed guest path segment.

(b) The blue path segment shows the connectivity of newly inserted path segment.

Figure 4.10: Insertion of two-way directed guest path segment into 1-way directed host path segments of periodic route.
CHAPTER 4. INITIAL SOLUTION GENERATION

The rest of the figures in 4.11, 4.12, and 4.13 is not doing any thing extra than what we have already discussed, but they are just a demonstration of the scenarios that we may encounter and take care of. And it is apparent that the direction of the guest path segments effect the way we should calculate our shortest path.

(a) Calculation of shortest path from each node of the host (b) After successful insertion, the blue path segment shows path segment to the start node of one-way directed guest the connectivity of the newly inserted path segment path segment

Figure 4.11: Insertion of a one-way directed guest path segment into one-way directed host path segments in periodic route

We may also come up with some situations while doing the insertion. These are grouped in figure 4.14:

After repeated number of insertion operation we may encounter scenarios as shown in figure 4.14, where we are trying to insert the very same edge inside a periodic route path segments resulting a redundant path segments. We have either a one-way or two-way path segments between node 3 and 4. In the top-most left figure we have a two-way directed host path segments and a two-way directed guest path segments. We are trying to insert the same edge between node 3 and 4, but with different status, so instead of inserting the guest path segment we just change the status of that edge in the host path segment. That helps us filter out redundant transportation path segment while inserting any path segment.

4.5 Swap Operation of Periodic Route Path Segments

Simulated annealing will be doing quite a few swap operations between periodic routes to optimize the path segments within our initial solution. The swap operation operates as follows:

- We choose the first periodic route $pr1$ in a random manner.
  - Then we choose a plough path segment in a random manner from the first randomly chosen periodic route.
4.5. SWAP OPERATION OF PERIODIC ROUTE PATH SEGMENTS

(a) Calculation of shortest path from each node of mixed-way host path segment to the start node of one-way guest path segment

(b) The blue path segment shows the connectivity of the newly inserted path segment

Figure 4.12: Insertion of one-way directed guest path segment into mixed way directed host path segments of periodic route

(a) Calculation of shortest path from each node of mixed-way directed host path segment to the start node of two-way directed guest path segment

(b) After successful insertion, blue path segment shows the connectivity of the newly inserted path segment

Figure 4.13: Insertion of 2-way directed guest path segment into mixed way directed host path segments of periodic route
Figure 4.14: Cleanup operation while insertion
4.6 Split Operation Periodic Route Path Segments:

- We choose another periodic route \( pr2 \) in random manner. While choosing the second one we make sure that we do not choose the one that is already chosen first time.
  - Then we choose a plough path segment in a random manner from the second randomly chosen periodic route.

- We remove the plough path segment from each of the periodic route with the procedure we have discussed in section 4.3.

- Then we try to insert the removed path segments from \( pr1 \) and \( pr2 \) to \( pr2 \) and \( pr1 \) respectively according to the process described in section 4.4.

- If the insertion is successful only then the swap operation is successful.

- Upon successful swap operation we update the time and cost for the newly updated periodic route.

4.6 Split Operation Periodic Route Path Segments:

The split operation filter out the plough path segments from the periodic route. We select some specific periodic routes whose plough path segments will be splitted based on some \textit{split factor constant} which are pre-defined and could be any fractional value between 0 and 1. The process is as follows:

\begin{algorithm}
\caption{Search operation of the periodic route whose plough path segments are eligible to be splitted}

\begin{algorithmic}
\While{the end of periodic route list is not reached}
\State Get the total time of the periodic route.
\State Get the period of the periodic route.
\If{(total time of periodic route < split factor constant * period of the periodic route)}
\State Store the location of the periodic route whose path segments will be splitted.
\EndIf
\EndWhile
\end{algorithmic}
\end{algorithm}

The algorithm 4.2 may find one or several periodic route(s) eligible enough to get their plough path segments splitted. If there is only one periodic route found we filter out the plough path segments, but if there are several, then we choose the one that has the highest transportation time of all selected. After the split operation we get a container as figure 4.15 of plough path segment organized according to similar time window.

4.7 Merge Operation of Periodic Route Path Segments

Merge operation tries to insert the plough path segments that are splitted into another periodic route if the periodic time constraint is not violated. Otherwise a new periodic route is instantiated with those splitted path segments. The resulted container of the split operation is browsed through and each element of the container contains one or several periodic route path segments with similar time window. Merge operation tries to insert these path segments into another periodic route if
the time constraint of those respective periodic route is not violated. If it is not possible to insert any of the path segments within the container into another periodic route, we have to create a new periodic route and insert those path segments into the newly created periodic route. The insertion procedure is same as described in section 4.4.

4.8 Cleanup Operation of Redundant Transport Paths

SA calls swap, split and merge operation in a repeated manner to make the initial solution optimized in terms of time and cost. While calling these functions the periodic route pile up quite few redundant transportation paths. Cleaning up the redundant transportation paths from the periodic route is also part of the optimization technique. There is no point of starting a periodic route with transportation since our main focus of optimization is to have as many as plough edges possible in a periodic route than transport edges. We can come up with different scenarios from where we can remove the redundant transportation paths.

First we address one-way directed periodic route that starts with transportation edges and mention them as redundant transportation paths which can be removed to reduce the time and cost of the periodic route.

The figure 4.16a is a one-way directed path segments starting with transportation and end with only plough path. We can clean up these transportation paths and calculate the shortest path from the end node of the plough path to the start node of the plough path. The figure 4.16b also starts with transportation path segments but we encounter plough path which is definitely not the end of the path. In that case we also remove one-way directed transportation paths from the beginning until we encounter a plough path segment and calculate the shortest path from the end node of the path segments to the start node of the first node of the plough path. The procedure that we have discussed so far is immediately followed by a procedure that removes transportation path from the trail of the periodic route path segments. The scenarios that we encounter here are shown in figure 4.17:

The discussion depicted in figure 4.16 and 4.17 was about cleaning up the redundant transportation paths from periodic route path segments consisting of only one-way directed edges. Now the following discussion will provide some insight of cleaning redundant transportation paths from two-way directed path segments. The steps that we follow are:
4.8. CLEANUP OPERATION OF REDUNDANT TRANSPORT PATHS

(a) Periodic route path starting with transportation path and the last one is the plough path
(b) Periodic route path starting with transportation and the last path is either plough/transportation path

Figure 4.16: Removal of 1-way directed transportation paths at the beginning of the periodic route paths

(a) Periodic route path ending with transportation and the first one is the plough path
(b) Periodic route path ending with transportation and the first plough path is not the first path in the route

Figure 4.17: Removal of 1-way directed transportation paths at the trailing of periodic route paths
Filter out all plough paths from the two-way directed periodic route path segments and store them in a list.

Filter out all transport paths from the two-way directed periodic route path segments and store them in another list.

Now browse through the main periodic route path segments list and check against the plough path list that we have stored in the first step. If there is any transportation path with the same edge we remove that transportation path from the main periodic route path segments since we already have an edge with plough status and we do not want to have the very same edge with transport status.

Then we check if there is any multiple transport path segment with the same edge index against the list we have created in the second step. If we find any multiple transport paths we remove one of them from the main periodic route path list.

### 4.9 Algorithm work-flow

In the previous sections we have described all the algorithms that will be used by simulated annealing after generating initial solution. The activity diagram in figure 4.18 will show how SA algorithm is functioning after the generation of the initial solution.

![Simulated Annealing activity diagram](image)

Figure 4.18: Simulated Annealing activity diagram

After the generation of initial solution we send this solution to the Simulated Annealing algorithm provided by Optiplan AB and it extracts the solution’s objective value and calls algorithms discussed in the previous sections and generates an optimized solution containing the periodic paths of the underlying network. The objective values are calculated from the following important and related attributes of edges and vehicles:

- **Vehicle**
  - Salary of the driver.
  - Vehicle competence - if the vehicle can plough and salt or can only plough.
  - Velocity for transportation.
  - Velocity for ploughing.
  - Different Costs
    - Fixed cost per hour.
    - Ploughing cost per hour.
    - Transportation cost per hour.
    - Salting cost per hour.
    - Waiting cost per hour.
    - Penalty cost per hour.

- **Edge**
  - Edge length.
  - Edge time window which the maximum allowed time that can be taken to plough an edge.
Chapter 5

Implementation and Results

This chapter is the implementation and evaluation phase of the algorithm we discussed in previous chapters. The use-case diagram gives a high-level overview of how the user interact with the system which is followed by requirement analysis and detailed class diagrams of the software system that we tried to implement. In this thesis, we have followed the evolutionary life-cycle model of development. For example, we produce the first version of the artifact, then we revised it and produce the second version and so on. The process can be shown in figure 5.1.

There were several options to represent the network and eventually we were motivated to use the adjacency method to represent the network to save time and space. The chapter ends with the result of the simulation process.

![Evolution-tree life-cycle model](image)

**Figure 5.1: Evolution-tree life-cycle model**

We can provide a brief overview of this project process work-flow from figure 5.1. During the first episode we came up with a solution which was not good enough to be accepted by Optiplan AB. The parameter of acceptance was not quantified because of the meta-heuristic nature of the problem and instead relied on their extensive experience in the field of study. So we had to instantiate a second episode where we implemented a new algorithm to get a better result with the very same underlying class design.
CHAPTER 5. IMPLEMENTATION AND RESULTS

5.1 Software requirement analysis

A software requirement is a property that must be exhibited in order to solve a real-world problem [SWE]. If the activities related to it are poorly performed, any software engineering projects become critically vulnerable. In SRRPTWDS, the product requirement is to generate reduced number of periodic paths of the network for snow removal. And to achieve it, we used C++ programming language and its STL as one of our main process parameters.

5.1.1 Functional requirement

Functional requirements describe the functions that the software is to execute. In SRRPTWDS, we have to come up with reduced number of periodic paths of the whole network to reduce time and cost of snow-ploughing operation. And the high-level functional requirement is depicted in figure 5.2

![Diagram](image)

Figure 5.2: functional requirement of the system

In the first use case of figure 5.2, we create database from the network. The network connectivities of two districts came in the form of Microsoft excel sheets which is entered into the database through a GUI. The GUI was already developed in C# and it was used only for data entry and probably will be used for road-network and periodic route visualization in future. The scope of the project starts from the next two use cases. The first used case is brought up here for sake of involvement and completion.

5.1.1.1 Specific requirements

- **Required External Interfaces**

  - **User Interfaces** - The external users are the employees belonging the authority of road maintenance. The user can insert the data of the new operational districts or modify nodes within a district.

  - **Hardware Interfaces** - The external interface used for accessing the system is the personal computers. The PCs may be laptops and no Internet connection is required.

  - **Software Interfaces** - The software is only functional over Microsoft windows that support the .Net framework, because the GUI is developed with C#.

  - **Performance requirement** - The PCs should be at least with 1 GB of physical memory.
5.2 REQUIREMENTS ELICITATION

5.1.2 Nonfunctional requirement

One of the most important nonfunctional requirements is time performance. The simulation will be run only once in a year unless any changes within the operational network. Even after that, simulation-time should be reduced as much as possible. Some pre-calculation was recommended and observed to speed-up the computation.

5.2 Requirements elicitation

Requirement elicitation is concerned with where software requirements came from and procedure followed to collect them from the stakeholders.

5.2.1 Requirement sources

It deals with the source of software requirements and the process that software engineer use to collect them. It is fundamentally a human activity which is conducted by the requirement specialist at Optiplan AB.

5.2.2 Elicitation techniques

Once the requirements source has been identified as Vägverket, Optiplan AB started eliciting the requirements through a series of interview.

5.2.3 Requirements classification

In this project we have classified the requirements in the following dimensions:

- We have put most of the effort for the functional requirements.
- The functional requirements has been directly imposed by Vägverket.
- There is a requirement on the product which we are going to deliver - a solution that contains reduced periodic paths of the whole network. But there was no specific requirement from the stake-holder on the process followed and it came from the project manager.
- The higher priority was to come up with a solution and since the algorithm complexity was getting out of control at the end of project, the run-time constraints became to be of less importance in this project.

5.3 Software design

5.3.1 Software architectural design

It basically describes how SRRPTWDS software is decomposed and organized into components[SWE]. The following UML diagram provides a top-level structure and organization of the software that
identifies various components.

Figure 5.3: High-level architectural diagram

From the above diagram we can see the composition has been used where the network is mainly composed of nodes, edges, and vehicles. The network object contains all the node, edge and vehicle objects after reading the database.

5.3.2 Software detailed design

Detailed design describes in detail each of the components identified in the architectural design. We have detailed the static view of the top-level design in the following manner:

5.3.2.1 Class Design

Class diagrams are the most common and the most important view of the design that we created. The class diagrams will show the interrelationships between classes. They are designed to show all the pieces of our solution and eventually they should convey a sense of the system to be built at rest.

- **Entity Class** - Entity classes usually represent the data that we want to store. They also encompass logical entities. A logical entity is typically views or the result of heterogeneous queries to retrieve the data from the underlying database. Finding the logical entities is relatively easy because the relational database theory is pretty well-understood and relational databases comprise a significantly recurring repository for the different entities. We need entities for single tables and heterogeneous views comprised of multiple tables. From that point on, all the entities are modeled as classes. We shall be using <<entity>> stereotype for the following classes because they will be long-lived and persisting. They contain the business functionalities and any interactions with back-end systems are generally done through these classes.

In the next section we shall develop a design and implementation for one of the most important control classes for the network. Before we do this we need some way to represent the set of nodes, edges and vehicles covering the edges.
We used a representation of the network that associates each node with a set of adjacent nodes (neighbors). This model is memory-efficient, because it stores information for precisely the edges that actually belong to the network [Ford2002]. For each node, an element in the adjacency set is a pair consisting of destination node and edge. We give the *adjacency set* representation with weighted-edges in figure 5.4 [Ford2002]:

![Network representation with weights](image1)

![Data-structure representation of the corresponding network](image2)

**Figure 5.4: Network representation by means of Adjacency set**

In this project we have followed the adjacency set representation to represent the underlying network. In figure 5.4a, we have a weighted network and in figure 5.4b we have the corresponding representation. The squares under the label *Nodes* in figure 5.4b represent the nodes in the network and the divided rectangles under the label *Set of Neighbors* represent the neighbors of each node. For example node A has four neighbors and they are nodes B, C, and edges e1, e2. Even though it seems that A has 2 neighbors, each *Neighbor* class object encapsulates the node and the edge which will be clearer in the detailed class descriptions. All through this project we have used the following STL containers to store the entities while running the simulation.

- **Associative containers** - map, multimap, set, multiset.
- **Sequence containers** - vector, list.
- **Adapter containers** - stack, queue, priority_queue.

The *Network* class that we use in that system uses the adjacency-set representation for a network and the entity classes are as follows:

- **Node** - The *Node* class has few other members, but we discuss the ones that are important for the sake of brevity. It declares public data members and both a default constructor and a constructor that initializes the *nodeMapLoc* data member with the location of the node in the map, which is contained inside the *Network* object. The following list describes the most relevant and important attributes of the *Node* class.

  * *nodeName* - A string name of the node object.
  * *nodeId* - A template type id of the node object.
  * *nodeValue* - A double type data stored and used while calculating the *Dijkstra's shortest path* between node objects.
* `nodeOccupied` - A boolean flag that indicates if the node is included in the network or not.
* `nodeInDegree` - An integer variable that indicates the number of edges arrives to the node object.
* `nodeMapLoc` - The data member `nodeMapLoc`, is an iterator for a map element and this map is stored in the `Network` class.
* `nodeNeighbors` - Each node has an associated adjacency set containing all of its adjacent outgoing nodes and edges encapsulated as `Neighbor` objects.

![Node Class Diagram](image)

**Figure 5.5: Node Class Diagram**

The figure 5.6 shows how the `Node` objects are stored in the `Network` and their connectivity. The `Node` constructor and the default constructor initializes the `nodeMapLoc` data member. It is of type `map<T,int>::iterator` - a location to key-value pair, where `nodeId` is the key with template type and the value is the index of `Node` object into the vector container where the `Node` object is stored. The vector container is `networkNodeList` - a data member in `Network` class. The map `networkNodeMap` is a data container member of `Network` class and `nodeMapLoc` points to one of its element.

![Node object with reference to the Map location and the Adjacency set](image)

**Figure 5.6: Node object with reference to the Map location and the Adjacency set**

- **Edge** - The `Edge` class has the following public attributes as shown in figure 5.7:
  * `edgeId` - A template type id of the `Edge` object.
  * `edgeName` - A string type value of the `Edge` object.
  * `edgeStartNodeIndex` - An integer value that specifies the node index in the vector of `Node` objects which is stored in the `Network` class. This index indicates that the `Edge` object starts from a `Node` object.
  * `edgeEndNodeIndex` - An integer value which specifies the node index in the vector of `Node` objects which is stored in the `Network` class. This index indicates that the `Edge` object at the end of the `Node` object.
5.3. SOFTWARE DESIGN

- **edgeLength** - This data member represents the length of the edge.
- **edgeTimeWindow** - It signifies the maximum amount of allowed time that the vehicle can take to plough this edge.
- **edgeCurrentStatus** - An object of enumerated type `CurrentStatusOfEdge` which defines the current status of the `Edge` object as shown in figure 5.7.
- **edgeToDoThings** - Another object of enumerated type `ThingsToDoOverEdge` which defines the task to be performed over each edge segment.
- **edgeMapLoc** - The data member `edgeMapLoc`, an iterator for a map element and this map is stored in the `Network` class. The `edgeMapLoc` maintains the connectivity between the `networkEdgeMap` and `networkEdgeList` as shown in figure 5.8. Both of the data containers are member of `Network` class.

![Figure 5.7: Edge class diagram](image)

![Figure 5.8: Edge object with reference to the Map](image)

- **Vehicle** - The `Vehicle` class declares public data which resembles the attributes described in section 4.9. However, it is relevant to describe them along with variable names.

  - **vehicleId** - A template type id of the `Vehicle` object.
  - **vehicleName** - A string type member for the `Vehicle` object.
* vehicleVelocityPloughing - A double type variable containing the ploughing velocity of the Vehicle.

* vehicleVelocityTransportation - A double type variable which holds the transportation velocity of the Vehicle.

* vehicleFixedCostPerHour - A double type variable and holds fixed cost per hour of the Vehicle object.

* vehiclePloughingCostPerHour - A double type variable which holds the ploughing cost per hour of the Vehicle object.

* vehicleTransportationCostPerHour - A double type variable which contain the transportation cost per hour of the Vehicle object.

* vehicleSaltingCostPerHour - A double type variable which holds the cost of spreading salt per hour for the Vehicle object.

* vehicleWaitingCostPerHour - There is some waiting cost associated with every vehicle which are instantiated in the Vehicle constructor.

* vehicleCapability - A variable of an enumerated type VehicleCanDo defines the capability of the vehicle.

* vehicleMapLoc - Both the default constructor initializes the vehicleMapLoc data member with the location of the Vehicle object in the map of Network object. It maintains the connectivity in the very same manner as Node and Edge object does and it is shown in figure 5.10:

![Figure 5.9: Vehicle class diagram](image-url)
5.3. SOFTWARE DESIGN

• **Control class** - Control class acts on other classes. It represents the bridge between entity classes and boundary classes. Implementation of this class depends on the style we follow. Control class can manage how data are marshaled to presentation classes and how data are marshaled to other systems through boundary classes.

  - **Network** - In this problem domain, Network class is acting as a control class that arranges the data in a data-structure after reading from database. Network class is very heavy in structure. It arranges not only the data from the database, but also runs all the simulation to arrive at an optimized solution. It contains all of the other classes as members that will be used to hold data from the database.

  The Network class is the most sophisticated container in that project as shown in figure 5.11. The complexity derives from the fact that we need separate data structures to store the nodes, the edges, and vehicles that are read from the database. In addition, we need information as we need to visit nodes in the traversal of the network. This information is critical in implementing many of the network traversal algorithms. We describe the most important attributes that bind the Node, Edge and Vehicle objects. They are:

    * **networkName** - A string type variable that holds the name of the network.
    * **networkNodeList** - A STL vector sequence container for Node objects. When we try to insert a new Node object, we check the following constraints:
      - We extract the nodeId of the Node object and try to insert it inside the member networkNodeMap. If nodeId already exists, we find the integer index corresponding to the key nodeId and update the corresponding vector element with the new Node object.
      - If the nodeId is not found, we insert a new Node object at the end of the vector.
    * **networkNodeMap** - The map<T,int> container also gets inserted during a Node object insertion and the value field gets updated with the index of Node object that is stored in networkNodeList. The key field of the map gets the nodeId. The connectivity between them is shown in figure 5.6.

  Rest of the other members networkEdgeList, networkEdgeMap, networkVehicleList and networkVehicleMap maintain the connectivities between each other in the same manner as just described. The readers are requested to review the figures 5.8, and 5.10.

  - **Neighbor** - The Neighbor class provides a data type for each element in the adjacency set. The public data member includes the adjacent (destination) nodes and edges. As we shall see, destination is represented as integer indices into the vector networkNodeList that contains the node properties and into the vector networkEdgeList that contains the
edge properties. The Neighbor object maintains the connectivity between Edge objects and Node objects through the use of the indices. The Neighbor contains a constructor that initializes the data members, plus overloaded operators $<$ and $==$ that compare two neighbors. The operators allow a set to contain Neighbor objects. The figure 5.12 gives an overview of the class that we have just described:

- **VehicleEdgeConnectivity** - The VehicleEdgeConnectivity connects the Vehicle and Edge class with the fact that each edge is traversed by vehicles of various capabilities through plough or transportation. And based on the vehicle capability and edge length we have various combinations of VehicleEdgeConnectivity objects. To speed up the simulation we pre-calculated these information before running the main simulation and store them in a hash table. To avoid collision, we have concatenated vehicle capability type and edge index in such a way that we always get unique hash id for each VehicleEdgeConnectivity object in the hash table. The figure 5.13 shows some of the attributes of the class.

- **ShortestPathAttributesBetweenNodes** - This class is responsible for holding the shortest path and time between nodes of all combinations possible. While developing the framework, we have observed that calculating shortest path at run-time time using the Dijkstra's algorithm is time consuming and therefore we have decided to generate all the combinations between nodes, calculate the shortest path and time between them before running the main simulation and store them in a hash table for faster look-up. The attributes startNodeIndex and endNodeIndex are the node indices we use for shortest path algorithm and this path will be covered by a vehicle with certain capability which we have already discussed. We store the time in the attribute shortestPathTime.
basic class structure of `ShortestPathAttributesBetweenNodes` is shown in figure 5.14:

![Diagram](image)

Figure 5.14: ShortestPathAttributesBetweenNodes class

- **PeriodRoute** - This class represents the core of the simulation process. `PeriodRoute` object contains the path segments within time and competency constraints. Before we go further, we need to elaborate more on both the constraints.

  * **Time constraint** - As we have discussed in the previous chapters that we have to plough the edges during snowfall within a certain time period. So every time we are about to take a new edge we check if we can plough the edge within the time constraint.
  
  * **Competency constraint** - A edge can be covered based on the capability of the vehicle. In our simulation all the vehicles have the least functionality to transport, but some vehicles may both plough and salt the edge while others can only plough. And again some of the edges are meant only to be ploughed or both ploughed and transported or only to be transported. So when we encounter an edge, we assign the vehicle capable enough to cover that edge based on vehicle capability and edge’s requirement.

After the generation of the initial solution for the system, the rest of the simulation process starts working with the periodic route objects. We have defined a struct `PeriodRoutePathStatus` that contains the status of each edge in the period route. That struct contains the integer edge index to point to the `Edge` object in `networkEdgeList` and `status` states the current status of the edge in the period route. The figure 5.15 highlights some of the important attributes of a typical `PeriodRoute` object.

![Diagram](image)

Figure 5.15: PeriodRoute class
Solution - The most light-weight class in the whole design framework. It contains the list of PeriodRoute objects. It seems unnecessary to have any class like that, but we considered to have it in the design because we have to deal with different Solution objects and Simulated Annealing will find the best solution of all the Solution objects. Again the simulated annealing takes the reference to the solution object. The class diagram for the Solution class is shown in figure 5.16:

![Figure 5.16: Solution class](image)

5.4 Testing

To perform all the test we have used the Sioux Falls district because of less complexity. It contains only 24 nodes as shown in figure 5.17. Based on experience and intuition of the Supervisor at Optiplan AB, the following testing was performed:

- **Ad hoc testing** - Several log files were created after reading contents of the database to make sure that the data reading operations are done correctly. Those log files were then used to match against the data that are actually stored in the database. No automated testing toolkit were generated to do the matching. For example we ran the function to calculate the shortest path between a node pair and matched that manually on the white-board for each and every nodes in the network. Several log files were generated for each and every functionalities in the simulation process after running them independently at least for 500 iterations.

5.5 Exception Handling

Several types of exceptions can cause the program to crash. We have relied upon the conventional C++ exception handler and the following types of exceptions were handled:

- **Memory Allocation Exception** - It is thrown if dynamic memory memory allocation is not possible.

- **Range Exception** - It is thrown if algorithm tries to access element in the array that is out of range.

- **Network Exception** - It is thrown if neighboring node or edge indices are not found, which gives an indication that the connectivity between the STL map and vector was not maintained.

5.6 Timeframe

The project started with an initial time plan. But it could not be observed during the algorithm development and programming phase, because substantial design changes within algorithms were made to arrive to a good solution and as a result implementation phase was delayed.

- Approximately two weeks were spent for preliminary studies.
- Approximately four weeks were spent in requirement analysis.
- Class design took about one and half week.
5.7. RESULTS

- Algorithm design, implementation and testing phase took most of the time during this work as bad solution required us to change the design frequently. Approximately twelve weeks were spent in this phase.

- The rest of the time was spent with the result and report generation.

5.7 Results

We have run the simulation on the network of Sioux Falls district (test network) and it took 20 seconds to run the simulation including the following operation:

- Reading the database and generation of data-structure.

- Combination generation between nodes to calculate shortest paths.

- Initial solution generation.

- Optimization using SA.

The initial solution produced 19 period routes and the simulated annealing reduced the number of period routes to 4. The figure 5.17 is a snapshot of the graphical user interface that represents the Sioux Falls network:

![Figure 5.17: Sioux Falls Network representation](image)
The reduced period routes have the following format:

```
The period route is: The name of the route is: PR_5_1.5 --> 4 --> 6 --> 6
The period of the route is: 2
The transportation cost of the period route is: 156.332 kr.
The transportation time of the period route is: 0.28568 hours.
The number of transport edges in the period route is: 11
The ploughing cost of the period route is: 554.766 kr.
The ploughing time of the period route is: 1.16689 hours.
The number of ploughing edges in the period route is: 20
The waiting cost of the period route is: 244.256 kr.
The Fixed cost of the period route is: 551.301 kr.
The total cost of the period route is: 1609.35 kr.
The total time of the period route is: 1.40539 hours.
```

Figure 5.18: Period Route Representation of Sioux Falls

The network of Eskilstuna is much bigger compared to the network of Sioux Falls. When we ran the same simulation over that operation district it took 96.2208 minutes to complete. The initial solution came up with 74 period routes and the optimized solution reduced that to 16 period routes. That network is assigned with 22 vehicles and each period route is assigned with a vehicle. Since we have reduced the number of period routes to 16, we are saving 6 vehicles which costs about several millions SEK each year.
Chapter 6

Discussion

This chapter will bring up some of the aspects that were experienced while going through the thesis work. Much have been learned in the field of process optimization techniques, not only theoretically but also practically while implementing them.

6.1 Learning Experience

This thesis work gave ample opportunity to get a hands-on practice on optimization problems in real-life network. At the same time it was perceived that the good-looking mathematical equations are not so beautiful to implement as a functional algorithm. From problem understanding, analysis and solving to implementation, several issues were identified that went right, wrong and could have been better if it were done the other way.

6.1.1 Area of choice

It was definitely a good choice to do the thesis work in that area. Even though the problem statement does not fit very well within the domain of Software Engineering & Management, but the basic software engineering techniques align very well to solve that practical scenario that turned out to be quite a pressing issue in snow falling countries and specially in Sweden. According to Optiplan AB, process optimization is the field of study which has not been practiced so much in real world even though it has many applications within the society. It has been addressed from the point of scientific computing so far. But it can definitely build up to an application framework with rich components and this is where software engineering starts to play its role. This framework started to get a shape with the thesis work and many other usability issues will enrich it in future.

6.1.2 Applied optimization concept

From the very beginning of the project, it was realized that lack of understanding in applied optimization concept will affect the problem of the work. Longer hours of literature study did not help much since the type of the problem was something very new and according to Optiplan AB and by the time this thesis was written, it was the first time any one in Sweden has incorporated both one-way and both-way edge path segment of the network to optimize the periodic paths. It was always associated with longer discussions to address the problem in a feasible way both in terms of memory and time. Fortunately, with unconditional support from Optiplan AB, problem understanding became easier. Those valuable hours could have been saved if at least a ground-level course work regarding applied optimization concept were attended.
6.1.3 Data-structure and algorithm

It was felt each and every instant while working in this thesis work that at least one course about data-structure and algorithm was necessary before enrolling into it. Even though advanced programming in C++ did help much to carry along, it could have been complemented with the concept of algorithm design and data-structure. All these concepts were learnt in parallel while understanding the core concept of optimization. The good thing is that it was well-managed with some good literature study in this field and useful suggestion from different people. One of the main issues was that we have used two container to store each entity class; one STL vector and one STL map. Even though it looks like excessive memory consumption, we have made a trade-off between memory and time through this process. STL vector is extremely fast for direct access and STL map is very fast to find value with key-value pair. While loading Node, Edge and Vehicle, we store each of these objects in vector containers and corresponding map containers containing each of their identification value as a key and the index of the vector as value. So from the map we find the index of the vector element by key and the process is very fast comparing to search each and every element in the vector in a linear fashion. Calculating the Dijkstra’s shortest path between any two nodes at run-time turned out to be very time consuming and therefore we decided to pre-calculate all the possible combination between 353 nodes, generate shortest path between each of these nodes and store them in a hash-table. Then we noticed significant speed gain in our simulation.

6.1.4 Adhoc testing procedure

Optiplan AB does not use either any test generators that assist in development of test cases or any test execution frameworks that enable the execution of test cases in a controlled environment where the behavior of the object under test is observed. Much of the effort could have easier if organized unit testing were introduced. Instead of doing the manual testing of each and every function with a text file output, we could have introduced unit testing where several test case classes are written to test and validate the functionalities of those functions.

6.2 Future steps

If we are asked to start working on the second version of this application, we shall be stressing refactoring and testing techniques which will enable us to develop the framework through range of functionality throughout the software system. It shall bring a wide range of benefits in both the short-term and the long-term for the software itself and the organization that sponsor and support the software.

6.2.1 Improved Object-Oriented Generic Programming

Great efforts have been put to build scientific libraries dedicated to particular application domains and main issue is to manage the large number of data types involved in the given domain. An ideal algorithmic implementation should be general. It should be written once and process data in an abstract way. Moreover, it should be efficient. Some libraries have been using generic programming to address this problem. In this thesis we have addressed the issue of managing different types of data. But the descriptions on a set of requirements on a type that parametrizes an algorithmic implementation are only defined in the documentation. It can be managed to be explicit in the program by representing the concept by type; moreover, concept of inheritance is fully supported. Thus it supports better procedural overloading, which is of prime importance for libraries where each algorithm implementation can have numerous variations.

Another problem is that generic programming paradigm suffers from a lack of appropriate design patterns. Classical design patterns of Gamma et al. can be translated into this paradigm while handling polymorphism operation by parametric polymorphism. In this patterns, we can solve the method calls statically, because the inferior type of each object in generic programming is known
at compile-time. So we can preserve the modularity and component re-usability while avoiding the performance penalty due to their dynamic behavior, which is critical in numerical computing. This will result in a better design for object-oriented generic libraries. In our case we have to refactor to generic patterns to attain the status of a generic libraries.

6.2.2 Parallel Programming

It has been well-revealed that we used Dijkstra’s Shortest Path algorithm many times in SA. By the time this thesis was written, we observed tremendous boom of multi-core processors in the consumer market and realized that we can gain more in terms of speed and efficiency if we manage to do parallel implementation of Dijkstra’s algorithm. The type of shortest path we are dealing with in this thesis is of single source, which means that we calculate shortest path from one source to one destination. A simple criteria which divide the Dijkstra’s sequential single source shortest path algorithm into number of phases, such that the operations within a phase can be done in parallel[ParaDijk1998]. According to Crauser, et al. (1998), simulations show the applicability of their approach even on non-random network and our network is generated prior the simulation which remains constant all through process. Thus it is apparent that the version of Dijkstra’s shortest path we implemented is an ideal candidate for parallelism.
Chapter 7

Conclusion and Future Work

In this thesis work we have described and try to provide an optimized solution to SRRPTWDS. We have given a mathematical model of the problem. The problem involves the snow removal activities that take place during snowfall and time-windows define periodic method of coordinating the ploughing and salting operation.

The snow removal problem with time window is hard to solve and has been recognized to be a NP-hard class optimization problems. We have used two principle methods to generate initial solution - one for two-directed edges and other for one-way directed edges. We have paved that way to deal with the scenarios in real-life that SNRA would usually encounter while ploughing and salting. We have combined both way to remedy the weaknesses if they were used in an isolated manner. We have compared and eventually motivated to use SA to provide optimized solution of the initial solution. The optimized solution did improve the solution but the process took much time to provide this solution and this is where we have room for future improvements - specially redesigning the initial solution generation algorithm and exploiting the strength of parallel computing. Again the path segments that we have generated are mixed of both one-way and two-way directed edges and the path segments are not cleaned from redundant transportation. Even though we have made our effort to clean up these redundant transport edges as much as possible, we finally realized that this issue has to be addressed in another thesis work.

The scientific output of this work hopefully provides the Swedish National Road Administration with valuable support into the performance of the snow removal system currently in use. Additionally, our results can be used to design cost-efficient snow removal operations.
Bibliography


[Ana] Busetti, Franco(n.d), *Simulated Annealing*


Detta dokument hålls tillgängligt på Internet – eller dess framtida ersättare – under en längre tid från publiceringsdatum under förutsättning att inga extra-ordinära omständigheter uppstår.

Tillgång till dokumentet innebär tillstånd för var och en att läsa, ladda ner, skriva ut enstaka kopior för enskilt bruk och att använda det oförändrat för ickekommersiell forskning och för undervisning. Överföring av upphovsrätten vid en senare tidpunkt kan inte upphäva detta tillstånd. All annan användning av dokumentet kräver upphovsmannens medgivande. För att garantera äktheten, säkerheten och tillgängligheten finns det lösningar av teknisk och administrativ art.

Upphovsmannens ideella rätt innefattar rätt att bli nämnt som upphovsman i den omfattning som god sed kräver vid användning av dokumentet på ovan beskrivna sätt samt skydd mot att dokumentet ändras eller presenteras i sådan form eller i sådant sammanhang som är kränkande för upphovsmannens litterära eller konstnärliga anseende eller egenart.

För ytterligare information om Linköping University Electronic Press se förlagets hemsida http://www.ep.liu.se/