NTNU

Norwegian University of
Science and Technology

Exact and heuristic solution approaches for a routing problem found within snow plowing operations

Anders Holmen Gundersen

Magnus Johansen

Benjamin Steffen Kjær

Industrial Economics and Technology Management

Submission date: June 2017

Supervisor: Magnus Stålhane, IØT

Co-supervisor: Henrik Andersson, IØT

Norwegian University of Science and Technology

Department of Industrial Economics and Technology Management

Problem Description

The purpose of the thesis is to develop models and solution methods to determine effective routes for snow plowing vehicles in urban areas. We consider two homogeneous fleets of vehicles, respectively dedicated to service traffic lanes, and pedestrian paths. The fleets are interconnected with synchronization constraints, giving the problem a high degree of complexity. The cumulative efficiency of the routes is measured by the makespan (the length of the longest route, measured in time units), which in Norway is the main constraint contractors are bound to adhere to.

We consider both exact and heuristic solution approaches, which are described, implemented, and evaluated. Most effort is given to a column generation method and a hybrid genetic algorithm with diversity management.

Preface

This master's thesis concludes our Master of Science degrees at the Norwegian University of Science and Technology (NTNU). We specialize in Managerial Economics and Operations Research at the Department of Industrial Economics and Technology Management. The thesis is based on the work we carried out in the subject TIØ4500 - Managerial Economics and Operations Research, in the fall of 2016.

Our motivation for continuing with the problem is mainly due to the possible practical implications it can have for municipalities throughout Norway and other countries subject to snowy winters. There is no doubt that one can lower costs for the society as a whole, and increase safety and mobility for pedestrians and drivers if the right solution methods are developed and implemented.

We would like to thank Trondheim bydrift for introducing us to the difficulties of the planning and execution processes involved in snow plowing operations, and particularly the problem explored in this thesis. No less, would we like to express our deepest gratitude for the interesting discussions, ideas, and the critically important feedback provided by our supervisors, Associate Professor Magnus Stålhane and Professor Henrik Andersson at the Department of Industrial Economics at NTNU. Without you, and your interest in this problem, this would not have been possible.

Anders Holmen Gundersen, Magnus Johansen & Benjamin S. Kjær

Trondheim, June 2017

Abstract

The Multi-Fleet Arc Routing Problem with Operation Synchronization (MFARPOS) is based on a problem faced by Trondheim bydrift when planning for snow plowing operations. The objective of the problem is to design routes for the plowing vehicles such that a road network is serviced in the shortest possible time. Two factors make the MFARPOS special with respect to the existing literature. 1) The problem is defined for two homogeneous fleets of vehicles: one to service the traffic lanes, and one to service the sidewalks and other pedestrian paths. 2) As all plowing trucks push the snow to their right hand side, all lanes adjacent to a sidewalk must be serviced prior to the respective sidewalk. The latter is referred to as the *synchronization criterion*.

We present a compact and a decomposed mathematical model for the problem. The arc-flow formulation (compact model) is an intuitive approach that has been implemented as a mixed integer program in a project prior to this thesis. A path-flow formulation (decomposed model) is also provided in order to obtain improved root node solutions on more problem instances. It is implemented as a Branch-and-Price algorithm and extensively tested. Results indicate that a variation of this approach is paramount to the implementation of the arc-flow model on small instances with many synchronization constraints and medium sized vehicle fleets. However, due to the complexity of the problem, no exact solutions can be obtained on instances of practical size. Thus, a Hybrid Genetic Search with Diversity Control (HGSDC), based on the work by Vidal et al. (2012), is implemented. The framework has shown to yield very good results on a variety of vehicle routing problems.

Due to the nature of the MFARPOS, we propose a new scheme for the heuristic to balance the diversity of the population, and test it on instance sizes ranging from small to large (up to 123 nodes and 836 arcs), all with a significant amount of synchronization constraints. Our results indicate that the heuristic is capable of finding optimal solutions to small instances in a short amount of time. On larger instances, where neither the optimal nor root node solution are known, the genetic algorithm is capable of generating solutions (although with some spread in makespan when run several times). More research need to be conducted in order to validate the performance of the heuristic. However, we note that this is the first known implementation of this framework for any arc routing problem, and that the results are promising.

As both the problem, and the solution methodology proposed, are new with respect to the related literature, our focus has been on the technical aspects. Therefore, a tool for instance generation has been developed in order to test our models thoroughly.

Sammendrag

MFARPOS (The Multi-Fleet Arc Routing Problem with Operation Synchronization) er basert på et problem hos Trondheim bydrift ved planlegging av snøbrøyteoperasjoner. Målet er å planlegge ruter for brøytebilene slik at et veinett blir brøytet på kortest mulig tid. To faktorer gjør MFARPOS spesiell med tanke på eksisterende litteratur: 1) problemet er definert med to homogene flåter: én flåte til å brøyte veiene, og én flåte som brøyter fortau og andre stier for fotgjengere. 2) Ettersom brøytebilene skufler snøen til høyre, må alle veibaner ved siden av et fortau brøytes før det respektive fortauet. Denne siste sammenhengen kan på norsk kalles for *synkroniseringskravet*.

Vi presenterer én kompakt og én dekomponert matematisk modell for problemet. Arc-flow-formuleringen (den kompakte) er en intuitiv fremgangsmåte som ble utviklet og implementert som et blandet heltallsprogram (MIP) i et tidligere prosjekt. Path-flow-formuleringen (den dekomponerte) er utviklet i denne hovedoppgaven med et mål om å oppnå bedre rotnodeløsninger på flere probleminstanser. Den er implementert som en Branch-and-Price-algoritme, og grundig testet. Resultatene indikerer at en variant av denne fremgangsmåten er bedre enn MIP-modellen på små instanser med mange synkroniseringskrav og middels størrelse på flåtene av biler. Derimot kan vi legge til at på grunn av kompleksiteten til problemet, er det ikke mulig å oppnå optimale løsninger på instanser av praktisk størrelse. Derfor har en genetisk algritme (HGSDC) blitt implementert. Rammeverket vi har basert algoritmen på har vist seg å fungere godt for mange problemer presentert i litteraturen som omhandler VRP (vehicle routing problem).

Grunnet strukturen til MFARPOS foreslår vi en ny fremgangsmåte for heuristikken for å balansere spredningen til populasjonen av løsninger, og tester den på små til store instanser (opp til 123 noder og 836 buer), alle med en signifikant mengde synkroniseringskrav. Resultatene indikerer at heuristikken er kapabel til raskt å finne optimale løsninger på små problemer. På større instanser derimot, der verken optimale- eller rotnodeløsninger finnes, er den genetiske algoritmen i stand til å generere løsninger (dog med en viss spredning i korteste rutetid når heuristikken kjører flere ganger). Vi konkluderer med at mer forskning må til for å validere kvaliteten på heuristikken. Uansett understreker vi at dette er den første kjente implementasjonen av dette rammeverket for noe ARP (arc routing problem), og at resultatene virker lovende.

Siden både problemet og løsningsmetodene vi har presentert er nye med hensyn til litteraturen, har vårt fokus vært på tekniske aspekter. Derfor har vi også implementert en instansgenerator for å kunne teste modellene grundig.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Each winter, Trondheim has an average of 27 snowy days (Pedersen, 2013). During or immediately after a snowfall, a fleet of snow plowing vehicles is deployed to clear the roads, bicycle paths and sidewalks. In Trondheim, there are 90 large plowing trucks dedicated to service the roughly 860 kilometers of municipal roads (Trondheim Kommune, 2016). In addition, a fleet of smaller vehicles is dedicated to narrower paths, such as sidewalks, pathways through parks, and other roads that cannot accommodate the weight or width of the larger trucks. The planning of operations, and the decision to deploy the assets are taken centrally by Trondheim bydrift, although the trucks are located at depots throughout the city. During the snowy months, these trucks are at all times on stand-by, ready to be called on duty. The rest of the year, the trucks serve other purposes. Most of the smaller vehicles are also multi-purpose vehicles.

When planning their snow plowing operations, Trondheim bydrift's paramount objective is to assign routes to all of the vehicles at their disposal, such that the road network is cleared within a given time limit decided by the municipality. That is, to decide when a vehicle shall plow which road, path or sidewalk. Today, this planning is performed centrally "by hand", in the sense that a person makes minor or major adjustments to the routes used the previous year. This is mainly to accommodate changes in the underlying road network, e.g. new roads, and changes in the vehicle fleet. According to Haarberg and Aleksandersen (2016), whose main responsibility is winter road maintenance in Trondheim, the drivers of the vehicles are only given a list of road segments to service. Thus, there exist no information regarding the order in which the roads shall be serviced. Additionally, most of the routes used today are not connected in such a way that the vehicles can plow continuously from start to end. Therefore,

throughout the workday, the driver has to make several decisions on how to get from one road segment in the route to the next. This notion of driving without plowing is referred to as *deadheading*. Trondheim bydrift emphasizes that it would ease the driver's job, and make the overall plowing operations more efficient, if complete routes with all information regarding when to service and deadhead which roads existed. However, in order to design such routes, many underlying requirements must be accounted for, and the right optimization tools must be implemented.

Within the operations research literature, problems concerned with a road network, where requirements are associated with the roads, are usually modeled as Arc Routing Problems (ARPs). In these problems, a graph is constructed to mimic the underlying road network. The arcs or edges in the graph generally represent roads, or parts of a road, and the vertices represent intersections. In problems of practical matter, the graph for which the mathematical problem is defined, is usually manipulated to represent changes on a road and ease the process of route generation. However, the core of ARPs - and what distinguishes them from the thoroughly studied Vehicle Routing Problem (VRP) - is that demand in the graph is associated with the arcs and edges, contrary to the nodes. As snow plowing is mainly concerned with service demand on roads, an ARP formulation is a natural starting point for an optimization model of the problem Trondheim bydrift faces.

In this thesis we investigate a new variation of the general ARP, namely the Multi-Fleet Arc Routing Problem with Operations Synchronization (MFARPOS), which arises when planning for snow plowing operations for the residential and urban areas of the City of Trondheim. Within the periphery of the city center, especially, not only cars and other motorized vehicles are affected by snowfall; the accumulated snow on sidewalks and pedestrian paths also has to be taken care of for safe travel. A specific problem arises for such paths (e.g. sidewalks) that are located adjacent to a road. When a vehicle is plowing, it pushes the snow to the right hand side of its direction of traversal, as shown in Figure 1.1. This implies that the leftmost lane needs to be plowed first when servicing road segments with multiple lanes in the same direction. When servicing the rightmost lane, the snow is pushed onto the sidewalk. Therefore, in order not to plow the sidewalk twice, it must always be serviced after the adjacent traffic lane, which in turn means that a sidewalk associated with one or more lanes will have to be serviced after all of these lanes. Figure 1.2 shows a vehicle plowing the sidewalk after the adjacent lane has been serviced. This relation, which we refer to as the *operation synchronization* criterion, is the underlying constraint that distinguishes the MFARPOS from other

Figure 1.1: A snow plowing truck, plowing the outer of two lanes in the same direction. Photo: FEMA/ Michael Rieger, desaturated from original.

Figure 1.2: A smaller vehicle is plowing the sidewalk adjacent to a lane. Photo: Jim Peaco (www.flickr.com), used under CC BY / desaturated from original.

ARPs studied to date.

In the context of the MFARPOS, a route is heavily dependent on the other routes spanning the network. Therefore, we argue, all routes within a constrained area should be constructed together, while accounting for traffic lanes, sidewalks and other pedestrian paths simultaneously. Methods to accommodate this will allow planners to free up assets from excess equipment and personnel, and/or plow the road network more efficiently; the practical implication being lower overall costs, better and safer driving conditions, improved mobility, and fewer accidents.

To the best of our knowledge, the MFARPOS is a so far unattended problem in the published literature. However, there have been some attempts to solve both general and highly specific problems related to snow plowing. Due to the complexity of these problems, the most promising solution methodologies to be used on realistic instances are based on heuristics. Alongside the evolution of heuristics and access to more computational power, the last decades of research have shown that many of these problems can and should be solved with operations research tools. Many arc routing problems can be broken down in the same manner, and most of the best heuristics have many commonalities. Vehicle routing problems, on the other hand, are far more studied; and a wider spectrum of solution methods have been developed for these problems. Among the successful solution approaches for larger instances are methods involving column generation and/or genetic algorithms. Their counterparts within the field of arc routing are close to non-existing. However, the research conducted within column generation and genetic algorithm for ARPs have shown promising results.

Computerized optimization tools will allow Trondheim bydrift to better plan their

snow plowing operations. However, efficient optimization models that produce high quality routes need to be developed prior to practical implementation. The purpose of this thesis is to develop and test better exact models and heuristics in order to solve larger instances of the MFARPOS. We note that a preliminary study of this problem has been conducted prior to the thesis. In Gundersen, Johansen, and Kjær (2016), we developed and implemented a mixed integer program (MIP) alongside a construction heuristic to solve for larger instances. We continue the development of exact solution methods by proposing a column generation approach in order to solve larger instances to optimality, and provide a better bound for even larger instances. To generate routes for the largest instances, we develop a hybrid genetic algorithm, inspired by state of the art methods from the literature on vehicle routing problems. Additionally, as there exist no benchmark instance data for the MFARPOS, we also develop a program for the purpose of designing instances based on characteristics of real road networks.

The outline of the thesis is as follows. In Chapter 2, we review the relevant literature associated with problems related to the MFARPOS. We introduce column generation- and genetic algorithm approaches, and place our work among the existing literature. A detailed description of the problem is provided in Chapter 3. The general mathematical model is given in Chapter 4, succeeded by a column generation reformulation in Chapter 5. In Chapter 6 we give an in-depth walkthrough of the genetic algorithm developed for the MFARPOS, followed by a description of our instance generation tool in Chapter 7. Results and discussion regarding our computational study are provided in Chapter 8. The thesis is concluded in Chapter 9, where we state our concluding remarks and give suggestions for further research on the topic.

Chapter 2

Literature Review

In this chapter, we present a brief overview of the existing literature relating to the MFARPOS. In Section 2.1, we introduce neighboring variations of the MFARPOS, along with their most significant solution methodologies. Special emphasis is given to problems concerned with snow plowing. Models where the problems are reformulated and solved through column generation are discussed in Section 2.2, while Section 2.3 is concerned with heuristic algorithms and their application within related problems. The chapter is concluded in Section 2.4, were we discuss our contribution. The reader should bear in mind that in this chapter, and throughout the thesis, *nodes* and *vertices* are used interchangeably.

2.1 Related arc routing problems

The basis of all variations of ARPs is recognizing that many problems are easiest modeled by associating demand with the arcs and edges in the graph, contrary to the nodes (such as in the VRP). Euler (1741) is credited for the first mathematical formulation of the ARP. In that problem, the bridges in the town of Königsberg are described as edges, and likewise the land areas are described as vertices. The goal is to find a single route from a starting point, that traverses all bridges exactly once (the demand), such that the traveler ends up in the initial land area. Since then, many variations of the Bridges of Königsberg problem have been formulated, often adding complexity to previous works. The Capacitated Arc Routing Problem (CARP), first introduced by Golden and Wong (1981), form the basis for most of the problems concerned with snow plowing. It is defined as the problem where K vehicles, all with a predefined capacity, Q, and based at

the same depot, must encompass a set of required edges. Several objective functions are interesting for the CARP. However, the makespan-objective, which is to minimize the length of the longest route, is the most commonly used. As pointed out by Benavent et al. (1990), the general case of the CARP is $\mathcal{NP}$-hard, although some simplified variants that can be solved in polynomial time exists.

Perrier, Langevin, and Amaya (2008) are among the first to describe algorithms that can handle instances of practical size, for applications within snow plowing. They develop two constructive heuristics for route generation: one that constructs routes for all vehicles in parallel, and one that split the graph into clusters before assigning one vehicle to each cluster. A central feature of their problem description is the hierarchical structure of the graph, commonly denoted as the Hierarchical Chinese Postman Problem (here with multiple vehicles). The edges in the graph are split into ordered subsets, such that all edges higher in the hierarchy must be serviced before lower ranked edges can be traversed. Additionally, their models can accommodate turn restrictions (such as forbidding U-turns) and tandem service needs, which are often a requirement on highways. Both of their models produce solutions that outperform the current routes used in the City of Dieppe, Brunswick, Canada.

Kinable, Hoeve, and Smith (2016) develop three optimization models to solve an extension of the CARP, where the fleet is heterogeneous - that is, the capacity of the vehicles may be different from one another. The models can therefore account for vehicles that need to resupply fuel at the depot, and are easily adjusted to the routing of spreading vehicles as well, such as carriers of salt and sand. The best performing model uses a constructive heuristic to find initial solutions, and improves upon these solutions by the use of two local search operators.

Although the aforementioned models perform well on the problems they are designed to solve, many problems concerned with snow plowing have been solved efficiently by using an Adaptive Large Neighborhood Search metaheuristic (ALNS). As described by Ropke and Pisinger (2006), the ALNS searches through a larger neighborhood than commonly known local searches, and choose operators (such as swap and switch) partially based on their previous success. The adaptive choices of operators make the ALNS a diverse heuristic, and is the main reason for its broad success. Within snow plowing operations, Salazar-Aguilar, Langevin, and Laporte (2011) introduce an ALNS heuristic to a problem where routes need to be synchronized in order for two or more vehicles to service multiple laned roads simultaneously. The authors develop and evaluate five destroy/repair operators and conclude that their individual success

is heavily dependent on the instance size. The heuristic is further evaluated on larger instances by Salazar-Aguilar, Langevin, and Laporte (2012). Additionally, it proves to generate realistic routes on a "difficult real example" (Salazar-Aguilar, Langevin, and Laporte, 2012, p. 1440). The ALNS framework also yields promising results on a variety of other problems, such as the Synchronized Arc and Node Routing Problem, as described by Salazar-Aguilar, Langevin, and Laporte (2013), and the Periodic Capacitated Arc Routing Problem with Inventory Constraints (Riquelme-Rodríguez, Langevin, and Gamache, 2014). Although snow plowing problems generally are modeled as deterministic, a closely related extension is to describe the amount of snow, and thereby the time demand on arcs or capacity of vehicles, as stochastic. This relates the problem to other stochastic problems, such as garbage collection, where the amount of garbage at each household is uncertain. Laporte, Musmanno, and Vocaturo (2010) implement an ALNS heuristic on such a problem, concluding that it is superior to other models.

For further reference to problems concerned with snow plowing and road maintenance, we recommend the four-part survey by Perrier, Langevin, and Campbell (2006).

2.2 Graph transformation and column generation

Due to its many applications, vehicle routing problems have been extensively studied in the literature, and thus many solution methods have been developed both with respect to exact models and heuristics. Its counter-class, the set of ARPs, has been given much less attention. For many problems concerned with arc routing, it has therefore been natural to reformulate the problems as a VRP (for many applications a Traveling Salesman Problem), and use known solution methods to solve them. Although the problem size of the VRP cast is often larger than the original ARP, the effectiveness of the solution methods have allowed for larger original problem instances. Laporte (1997) provides a general formulation for casting several classes of arc routing problems as TSP. A computational study is carried out, concluding that the method works well on low density graphs; that is, graphs with few edges relative to vertices. However, for some ARPs, e.g. the Mixed Rural Postman Problem and the Stacker Crane Problem, as defined by Eiselt, Gendreau, and Laporte (1995), this kind of graph transformation constitutes the only known approach where optimality can be proven. Dror and Langevin (1997) and Blais and Laporte (2003) propose similar graph transformations for the Clustered Rural Postman Problem and the Generalized Routing Problem, respectively, both yielding competitive computational results.

Thanks to the last two decades of research on arc routing problems, it is generally understood that more and more problems can just as efficiently be solved directly - without graph transformations. This is exemplified by Clossey, Laporte, and Soriano (2001), who describe a direct way to handle ARPs with turn penalties, which previously were handled through a VRP cast. As pointed out by Feillet, Dejax, and Gendreau (2005), such transformations are costly in terms of the graph size, and should as a rule of thumb be avoided. However, to avoid them altogether requires further research on many variants of the ARP. For the problem studied in this thesis, the MFARPOS, such a graph transformation was attempted (Gundersen, Johansen, and Kjær, 2016). The results were not promising, due to the increased size of the transformed graph.

Inspired by the success of column generation methods (as explained by Desaulniers, Desrosiers, and Solomon (2006)) to solve larger instances of various VRPs, Feillet, Dejax, and Gendreau (2005) examine this approach on the Profitable Arc Tour Problem (PATP). The problem stems from tactical freight transportation planning and has many commonalities with the aforementioned CARP. However, the PATP is a profit maximization problem, such that all demand need not to be accommodated. Additionally, there is no restriction on fleet size, and the fleet is not associated with a depot. To lessen the number of variables, the authors exchange the job assignment variables with binary variables associated with selection of elementary cycles in the graph. Elementary cycles are closed loops of arcs which can only occur once in the cycle. Note that restricting the cycles to elementary cycles in this problem will not restrict the possible solution space. The idea of using cycles is further transferred to the Branch-and-Price (B&P) procedure, called the flow-splitting method, where branching is conducted on short sequences of arcs instead of single arcs. Although the paper constitutes the initial research on the PATP, computational results are promising, showing that the column generation method is able to solve instances with up to 30 vertices and hundreds of arcs, which "...could not have been tackled by merely transforming the problem into a node-routing one" (Feillet, Dejax, and Gendreau, 2005, p. 551).

Letchford and Oukil (2009) argue that instances arising in practical applications of the CARP mostly are defined on sparse graphs, and that this sparsity should be exploited in pricing routines when solving the problem through a Branch-Cut-and-Price algorithm. As most graphs in ARPs originate from road networks where vertices correspond to intersections and edges to road segments, these sparse graphs are seen throughout most variations of this class of problems. Contrary to the methods presented by Feillet, Dejax, and Gendreau (2005), Letchford and Oukil (2009) argue that

pricing with non-elementary cycles can be completed in a reasonable amount of time on sparse graphs when using Dijkstra's Single-Source Shortest Path algorithm as a subroutine to speed up certain computations. A heuristic approach is nonetheless provided for larger CARP instances. Albeit a strongly $\mathcal{NP}$-hard problem, the authors also propose a method for pricing with elementary cycles, arguing that it should be formulated as a mixed-integer program, when the graph is sparse.

Of more recent work, Christiansen, Lysgaard, and Wøhlk (2009) formulate the CARP with stochastic demands (CARPSD) as a Set Partitioning Problem and solve it through a Branch-and-Price algorithm (without graph transformation). As the problem is of stochastic nature, two key elements arise that make the CARPSD different from other ARPs: 1) the objective is to find the minimum *expected* cost, and 2) the accumulated demand along a route might overcome the vehicle capacity. A feature of their solution method is the construction of shortest path edges between nodes, much like one would do in a graph transformation. However, the shortest paths generated in the CARPSD can both consist of deadheading and servicing. The problem description allows for this, as servicing an edge takes the same amount of time as deadheading it. The algorithm yields, in general, a very tight dual bound in the root node solution, although the tree size may become large.

Bode and Irnich (2012) present the first full-fledged B&P algorithm for the CARP. Their work utilizes many of the recent advances within operations research, such as symmetry elimination, efficient pricing, and branching on consecutive edges. A one-index formulation is used to strengthen the LP relaxation, and thus the lower bound. Although the column set in the master problem is significantly enlarged, non-elementary cycles are used in order to ease the pricing problem and keep the computational efforts low. Computational experiments underline the efficiency of the algorithm, where it is tested on several benchmark instances. The algorithm is used to solve the last open instance of the well-known benchmark set of Belenguer and Benavent (1998).

2.3 Heuristic algorithms for arc routing problems

Most arc routing problems related to the MFARPOS are very hard to solve, and the best exact solution methodologies generally fail to prove optimality when the graph size exceed 20-30 nodes (depending on the number of edges, problem type, etc.) Additionally, when the problem size increases even further, many models fail to even obtain a root node solution in a reasonable amount of time. This is a major problem, as graphs

that model road networks of practical size for planners, might surpass several hundred nodes. Therefore, the advent of heuristics has been essential in order to be able to use optimization tools as a decision support in the real world. The most popular heuristic framework used for snow plowing problems, namely the ALNS, has already been discussed. We now continue reviewing heuristic approaches, here focusing on genetic algorithms (GAs).

Although the popularity, and thereby the use, of GAs have increased significantly within many branches of computer science, few have attempted to solve arc routing problems with them. Those of significance to this thesis are all based on the CARP. These are discussed in the following, before concluding the section by summarizing some of the more interesting results from genetic algorithms used within vehicle routing problems.

Lacomme, Prins, and Ramdane-Chérif (2001) present the first GA published for the CARP. The algorithm is designed to account for several extensions, including prohibited/penalized turns, different costs for deadheading and servicing arcs, and directed data structures. A local search is included as a mutation operator on each child produced. The authors argue that "... it is clear nowadays that hybrid GAs are better ..." (Lacomme, Prins, and Ramdane-Chérif, 2001, p. 478), here referring to the algorithm as a *hybrid* due to the local search. Furthermore, they conclude that the algorithm is *very* efficient, as it performs at least as good as the best known algorithm at the time (a tabu search), on two sets of test instances, despite its simple form. Lacomme, Prins, and Ramdane-Cherif (2004) present a thorough study of several basic components that can be combined to hybrids GAs (called memetic algorithms in the paper) for the CARP. All of these components, which of there exist 12, are able to account for the aforementioned extensions. Unsurprisingly, the best combinations of these basic components perform very well. However, this is also true for the standard parameter settings used on all instances, despite the fairly simple methods each component of the GAs represent. Another hybrid GA is developed by Liu, Jiang, and Geng (2013) for the CARP. Special for their algorithm is the use of an iterated local search on a child solution dependant on its cost deviation from the best solution in the parent population: if the cost is close enough to the best solution in the parenting population, the local search is initiated. Although this increases intensification, several mechanisms, such as allowing infeasible solutions during the local search process, ensures a diverse population.

The makespan-objective that is generally used for the CARP is often criticized in the literature, and many authors suggest that alternative objectives may have a higher

value for practical purposes, although few present concrete alternatives. Lacomme, Prins, and Sevaux (2006) investigate the use of a hybrid GA for a specific, multi-objective extension of the CARP. The algorithm is used to develop a front between two objectives: the makespan and the total cost of the solution.

While genetic algorithms within arc routing mainly have been focused around the general CARP, their counterparts in the literature concerned with vehicle routing problems are far more extensive. Prins (2004) presents a hybrid GA, claiming that it is the first of its kind to be able to compete with the powerful tabu search methods designed for the VRP. Again, the author emphasizes that a local search is indeed needed to make the algorithm this strong. Early convergence that may be caused by the local search is countered by the use of small, distinct initial solutions, which are created by three classical heuristics.

We conclude this section with summary of the much referenced work by Vidal et al. (2012). They present an algorithmic framework called the *Hybrid Genetic Search with Adaptive Diversity Control*. The key feature of the heuristic is the combination of evolutionary algorithms, neighborhood-based metaheuristics, and advanced diversity control. The authors argue that the fitness of a solution should not only be determined from the cost, but also from a measure of the distance to the other solutions - which can be understood as the added diversity the specific solution brings to the total population. The algorithm is evaluated on three vehicle routing problems: the multi-depot VRP, the periodic VRP, and the multi-depot periodic VRP with capacitated vehicles and constrained route duration. On all of these problems, the algorithm outperforms all of the currently known metaheuristics on standard literature benchmark instances. The strong results of the algorithm is generally attributed the population-diversity management. It makes it possible to avoid premature population convergence, while leading to higher quality solutions in reduced computing time, compared to its competitors. The metaheuristic framework is expected to perform well on other variations of the VRP as well.

2.4 Our contribution

Problems related to snow plowing are highly diverse in the sense that almost no two papers investigate the same problem. New research is to a large extent dependent on what has been conducted before, and is often extending both the problem complexity and the quality of the solution methods. However, although many approaches have

been investigated, it now seems that the methods used to solve problems concerned with snow plowing somewhat have converged towards neighborhood search heuristics. Various implementations of the ALNS heuristic framework, first described by Ropke and Pisinger (2006), is commonly used on most variations of these problems. Two main reasons present themselves as to why this is so: 1) exact methods have so far been too weak to solve most problems of practical size close to optimality, and thus they are not further improved; and 2) the most promising heuristic approaches to many former studied problems have been based on the ALNS framework.

When reviewing research conducted on problems related to those concerned with snow plowing, and especially within the VRPs, one quickly finds that researchers have tackled these problems from many angles. For instance, the use of methods involving column generation (both exact programs and heuristics) have proven to be useful on a number of problems. This, however, is not the case for ARPs. Only a few attempts to use Branch-and-Price on arc routing problems exist in the literature, and only within the last decade has the CARP been solved with it (we refer to Bode and Irnich (2012) for details). The question as to why this is so may have many answers, but given the fairly modest published research on arc routing in general, we strongly believe that algorithms based on column generation can contribute to better solutions on ARPs. Drexl (2012) underlines the efficiency of column generation approaches to vehicle routing problems with multiple synchronization constraints - which are closely related to the problem studied in this thesis. Based on their success within VRPs, we also strongly believe that genetic algorithms are just beginning to show their strengths within arc routing. Among work of importance, we find Vidal et al. (2012), whose algorithmic framework, with its adaptive control of population diversity, yield competitive results on many variations of VRPs. Yet, it remains neither proved nor disproved whether a similar approach would yield good solutions on ARPs.

The Multi-Fleet Arc Routing Problem with Operation Synchronization is new with respect to the existing literature on arc routing problems, in which it fits neatly within the subset of problems associated with snow plowing. However, this is not to say that the mathematical problem formulation and the proposed methods for solving it, cannot be applied to describe and solve other practical problems. With respect to solution methods, we first implement the problem as a MIP program, based on prior studies. Additionally, it is reformulated as a Dantzig-Wolfe decomposition, and solved by a Branch-and-Price algorithm. This is implemented in order to obtain optimal solutions on larger data instances than the direct MIP implementation, improve bounds, and to

assess the value of such a formulation of this ARP. A hybrid genetic algorithm is developed and implemented to solve instances of practical size, although with the lack of guaranteeing optimality. Approaching arc routing problems with a column generation formulation and with the use of genetic algorithms is very unconventional. However, we strongly believe that these methods may prove their usefulness within arc routing, as they doubtlessly have on vehicle routing problems. Thus, with respect to solution methodology, our work is to a large extent new within arc routing. Lastly, since no benchmark instance data exist for this type of problem, an instance generator is also developed, and test data is provided.

For the purpose of a more specific classification with respect to problem characteristics, we have named the special features of the MFARPOS based on the problems discussed by Drexl (2012). In the survey, the author classifies many variants of the VRP that involve synchronization constraints, and discuss the solution methods that are commonly used to solve them. Our problem falls within the category of *Operations Synchronization*.

Chapter 3

Problem Description

In this thesis, we address the Multi-Fleet Arc Routing Problem with Operation Synchronization - an extension of the Capacitated Arc Routing Problem, found within snow plowing operations. It originates from the City of Trondheim, where every winter comes with heavy snowfalls that are challenging to accommodate from both a planning and operational point of view. This chapter provides a formal description of the MFARPOS, and addresses the issues that must be accounted for when modelling it.

The area under consideration for the MFARPOS is best described as a road network. A road network consist of a set of interconnected road segments. Each segment may have one or two directions, multiple lanes, and a sidewalk in either direction. The service demand of each road segment is obtained directly from the number of lanes and sidewalks. For example, a road segment with two lanes in each direction and two sidewalks (one on both sides), must be serviced a total of six times. A road segment, as we define it here, differs from a general road in the sense that the demand is constant everywhere on the segment. This is contrary to roads, where a lane might split into two, or a sidewalk ceases to exist midway. Additionally, a road segment has only two exits, one in each end. Therefore, it is not possible to traverse only half a segment. Note that it is possible for a road segment to only consist of one or two sidewalks, without having associated traffic lanes. Although these segments in practice often are pedestrian paths through parks, or narrow bridges where driving is prohibited or impossible, we use the term *sidewalk* whenever we refer to one of these paths, as they are serviced by the same vehicle type as the true sidewalks. Equally, we use the term *lanes* for the traffic lanes on a road segment. Locations where different road segments meet are referred to as *connections*. Intersections and roundabouts are typical connections in the road

Figure 3.1: Map of a residential area in Trondheim.

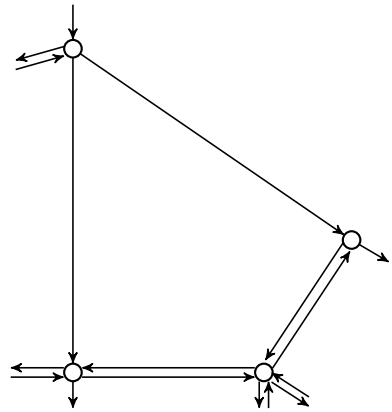

Figure 3.2: Road network of the residential area in Trondheim.

network. However, with our definition, these locations need not to be explicitly defined by a cross of multiple roads. Thus, the underlying road network for the problem studied has at least as many connections as the number of intersections. Figure 3.1 shows a small road network, found in downtown Trondheim. Figure 3.2 is a graph interpretation of the same network. Black arrows correspond to single lanes, where the arrowheads indicate the direction of the lane, whereas circles correspond to connections in the road network. Sidewalks are omitted from this illustration. With the aforementioned definition, all lanes (and sidewalks) between the same two connections collectively constitute a road segment.

The traversal times, both servicing and deadheading, for all lanes and sidewalks are known to the planner. The deadheading and service times for all lanes in the same direction on the same road segment is equal, respectively. However, deadheading a road segment always takes equal or shorter time than servicing it. A key feature of the problem under study is recognizing that all vehicles push the snow to their right hand side. Therefore, when a lane adjacent to a sidewalk is serviced, the snow will be pushed onto the sidewalk. If the sidewalk is serviced prior to the lane, it will have to be serviced again after it is spoiled with snow. Correspondingly, if a road segment has multiple lanes in the same direction, the snow on the leftmost lane will be pushed onto the lane to the right. Therefore, a paramount constraint when planning routes is to ensure that servicing always start in the middle of the road (the leftmost lane in each direction) and end with the sidewalk, if there is one. Sidewalks next to a lane have directions in order

not to push the snow onto the already serviced lane. Road segments only consisting of sidewalks may have a preferable side that the snow shall be pushed to. In this thesis, we assume that all sidewalks are to be modelled with a direction.

The vehicles used to plow the lanes are large trucks with an attached plow in the front that can be lowered and raised for servicing and deadheading, respectively. The width of the plow is of such a dimension that it is enough to cover the width of a lane, with a margin for error. A single traversal on a lane with the plow lowered will therefore always be enough to service the lane. However, the size and/or the weight of the trucks makes it impossible for them to service the sidewalks. These are serviced by smaller (and generally slower) vehicles, which cannot service the lanes due to the width and capacity of their plows. Within their respective fleet, all vehicles can be used interchangeably. It is therefore reasonable to model the fleets as homogeneous. Additionally, within the planning horizon of this problem, the size of the vehicle fleets are generally not subject to change. As a result, the number of available vehicles for service should be modeled constant.

Due to the size of the larger trucks, and the characteristics of the road network, it is not only problematic and time consuming, but often impossible for them to perform U-turns in connections. U-turns should therefore be prohibited for these trucks. On the other hand, the smaller vehicles can easily cross a road on pedestrian crossings, and turn around on small areas (such as a medium sized sidewalk). U-turns are therefore allowed for the smaller vehicles. Additionally, left hand turns may be time consuming and disruptive for the larger vehicles, and may cause them to leave a bank of snow in the middle of the road. Which turns that should include turn restrictions or impose extra waiting time for the trucks, is assumed known to the planner.

Common for all vehicles is the depot at which they begin and end their route. In the road network, this depot is associated with a single connection that represents one or two things in the underlying area: 1) the true depot location where the vehicles are located on their idle time, or 2) the location where the vehicles enter the road network if their true depot is not located in the area under study. In case of the latter, the connection modelling the depot is located at the edge of the road network.

Keeping a road network free from snow is a continuous process. Immediately after a snowfall, the main priority is to make the roads and sidewalks accessible for vehicles (buses, cars, motorcycles etc.) and pedestrians and cyclists. In Trondheim, the first response to a snowfall is therefore to service a set of predefined road segments. These are prioritized lanes and sidewalks, and never constitute all of the lanes and sidewalks

that eventually need to be cleared from snow. The set of prioritized lanes and sidewalks is what we refer to as the *road network*. Since most of the trucks used for the first response are owned by contractors, and service is often needed at unfortunate hours, this is where the majority of the costs lies. The lanes and sidewalks that are not serviced immediately after a snowfall are in most cases serviced during daytime, by vehicles owned by the municipality.

The aim of the problem studied in this thesis is to design routes for the first response, such that the road network is serviced in the least amount of time. This is often referred to as minimizing the makespan, and can be described as minimizing the time of the longest route. In the context of this problem, a route is a list of road segments that have to be serviced or deadheaded in chronological order. If any waiting time occur between two consecutive road segments for a vehicle, this information should be included in the route. One route shall be assigned to each vehicle. The routes must be designed in such a way that all the aforementioned criteria are satisfied. The set of all the routes shall provide the planner with information regarding the locations of all the vehicles at all times, and especially when the entire road network has been serviced and the last vehicle returned to the depot.

Chapter 4

Mathematical Model

In this chapter we introduce a compact mathematical formulation of the problem described in Chapter 3. The model is an arc-flow formulation, which we find to be the most intuitive way of formally presenting the problem under study. It is based on a mathematical model developed in the project leading to this master's thesis and presented in Section 4.1. A brief discussion regarding implementational improvements and simplifications concludes the chapter.

4.1 Arc-flow formulation

Let $\mathcal{G} = (\mathcal{V}, \mathcal{A})$ be a directed multi-graph where the vertex set $\mathcal{V}$ represents the connections in the road network (geographic locations with changes in service criteria), and the arc set $\mathcal{A}$ represents the lanes and sidewalks. If there is more than one category of service demand (both lanes and sidewalks) between two respective nodes, this is represented with two separate arcs, although belonging to the same road segment. If a segment has multiple lanes in the same direction, these are represented by one arc only. Note that all arcs are directed, such that there can be a maximum of four arcs between two nodes.

The set of vehicles $\mathcal{K}$, is separated into two fleets, $\mathcal{K}^L \cup \mathcal{K}^S = \mathcal{K}$. Let $\mathcal{K}^L$ be the set of plowing trucks for the lanes, and $\mathcal{K}^S$ be the set of vehicles that service the sidewalks. The trucks can only service and deadhead the lanes, while the smaller vehicles can service and deadhead the sidewalks and deadhead the lanes. Let $\mathcal{A}^S \subseteq \mathcal{A}$ represent the arcs that the vehicles for sidewalks can traverse, and let $\hat{\mathcal{A}}^S \subseteq \mathcal{A}^S$ be the set of arcs that have sidewalks that demand service. Similarly, let $\mathcal{A}^L \subseteq \mathcal{A}$ represent the arcs that can

be traversed by the plowing trucks, and let $\hat{\mathcal{A}}^L \subseteq \mathcal{A}^L$ be the set of lanes which have to be serviced.

In order to ease the mathematical formulation of the problem (and the implementation), a parameter T^{Max} is included in the description, such that in order to fulfill the service requirements, each vehicle $k \in \mathcal{K}$ has to drive a defined route within the time limit T^{Max}. This limit should be set sufficiently high, such that it is not a binding restriction on the makespan. Each route starts and ends at the same depot, D, and each traversal of an arc corresponds to a leg, n, in a route. Since a route can pass through the depot several times, we define the vertices $\hat{O}$ and $\hat{D}$ as artificial origin and destination depots, which are the nodes where all routes start and end, respectively. The artificial depots are only connected to the original depot D. The maximum number of arcs included in a route, often referred to as legs, is given by $\overline{n}$, and we define the set of possible legs, $\mathcal{N}$, as $\{1,...,\overline{n}\}$. As with T^{Max}, $\overline{n}$ should also be set sufficiently high. An arc can have a number of lanes or sidewalks in the same direction, and we define R^L_{ij} and R^S_{ij} to be the total number of lanes and sidewalks from node i to j, respectively. Let T_{kij} be the time vehicle k uses to service arc (i,j), and T^D_{kij} be the time vehicle k uses to deadhead arc (i,j), even if it is serviced or not.

Let the binary variables x_{kijn} be 1 if vehicle k service arc (i,j) on the nth leg of its route, and 0 otherwise. Similarly, let y_{kijn} be 1 if arc (i,j) is traversed by vehicle k and appears on the nth leg of the route while deadheading, and 0 otherwise. Variables τ_{kn} indicate the end time of service or traversal of leg n in vehicle k's route, while the variable t^{MS} tracks the makespan of the schedule.

Below follows the arc-flow model, which is a mixed integer program. For shorthand notation, we denote $\mathcal{A}_k$ as the set of arcs vehicle k can traverse, and for a given vehicle k the sets $\delta^+_k(i) = \{j|(i,j) \in \mathcal{A}_k\}$, and $\delta^-_k(i) = \{j|(j,i) \in \mathcal{A}_k\}$. Let M be a sufficiently large number.

Objective function

$$\text{Minimize} \quad t^{MS} \tag{4.1}$$

s.t.

$$\sum_{j \in \delta^+_k(\hat{O})} y_{k\hat{O}j1} = 1 \qquad k \in \mathcal{K} \tag{4.2}$$

$$\sum_{n \in \mathcal{N}} \sum_{j \in \delta^-_k(\hat{D})} y_{kj\hat{D}n} = 1 \qquad k \in \mathcal{K} \tag{4.3}$$

$$\sum_{j \in \delta_k^+(\hat{O})} \left(x_{k\hat{O}jn} + y_{k\hat{O}jn} \right) = 0 \qquad k \in \mathcal{K}, n \in \mathcal{N} \big| n > 1 \quad (4.4)$$

$$\sum_{n \in \mathcal{N}} \sum_{k \in \mathcal{K}^L} x_{kijn} = R_{ij}^L \qquad (i,j) \in \hat{\mathcal{A}}^L \quad (4.5)$$

$$\sum_{n \in \mathcal{N}} \sum_{k \in \mathcal{K}^S} x_{kijn} = R_{ij}^S \qquad (i,j) \in \hat{\mathcal{A}}^S \quad (4.6)$$

$$\sum_{i \in \delta_k^-(j)} \left(x_{kijn} + y_{kijn} \right) - \sum_{i \in \delta_k^+(j)} \left(x_{kji(n+1)} + y_{kji(n+1)} \right) = 0 \qquad k \in \mathcal{K}, j \in \mathcal{V}, \quad (4.7)$$

$$n \in \mathcal{N} \big| n < \overline{n}$$

$$\sum_{(i,j) \in \mathcal{A}_k} \left(x_{kijn} + y_{kijn} \right) \leq 1 \qquad k \in \mathcal{K}, n \in \mathcal{N} \quad (4.8)$$

$$\tau_{k(n-1)} + \sum_{(i,j) \in \mathcal{A}_k} \left(T_{kij} x_{kijn} + T_{kij}^D y_{kijn} \right) \leq \tau_{kn} \qquad k \in \mathcal{K}, n \in \mathcal{N} | n > 1 \quad (4.9)$$

$$\tau_{kn} \leq t^{MS} \qquad k \in \mathcal{K}, n \in \mathcal{N} \quad (4.10)$$

$$\tau_{kn} - M\left(1 - x_{kijn}\right) - T_{kij} \leq \tau_{k'n'} + M\left(1 - x_{k'ijn'}\right) - T_{k'ij} \qquad k \in \mathcal{K}^L, k' \in \mathcal{K}^S, \quad (4.11)$$

$$n, n' \in \mathcal{N},$$

$$(i,j) \in \hat{\mathcal{A}}^L \big| (i,j) \in \hat{\mathcal{A}}^S$$

$$x_{kijn} \in \{0,1\} \qquad k \in \mathcal{K}, (i,j) \in \mathcal{A}_k, \quad (4.12)$$

$$n \in \mathcal{N}$$

$$y_{kijn} \in \{0,1\} \qquad k \in \mathcal{K}, (i,j) \in \mathcal{A}_k, \quad (4.13)$$

$$n \in \mathcal{N}$$

$$\tau_{kn} \in \left[0, T^{Max}\right] \qquad k \in \mathcal{K}, n \in \mathcal{N} \quad (4.14)$$

$$t^{MS} \geq 0 \quad (4.15)$$

In the objective function (4.1) the makespan of the schedule is minimized. Equations (4.2) and (4.3) ensure that each route starts and ends in each vehicle's depot, while equations (4.4) provide that each vehicle can only leave the depot in the first leg. Further, equations (4.5) and (4.6) ensure that all arcs with demands are serviced, and constraints (4.7) make sure that the plowing routes are connected. Each vehicle can only traverse one arc in each leg of its route; this is taken care of by constraints (4.8). Constraints (4.9) state that the vehicles behave consistent according to time, and constraints (4.10) track the makespan of the schedule, while constraints (4.11) ensure that the synchronization requirements between the corresponding lanes and sidewalks hold. The remaining constraints are variable restrictions.

Big M

In constraints (4.11) an M-parameter is included, which we now derive in this section. The *big M* represents a sufficiently large number and is used to ensure feasible solutions for all allowed values of the variables. In order to tighten the relaxation of the arc-flow model, we wish to make M as small as possible. When $x_{kijn} = 0$ the left hand side should not exceed 0, and thereby $M = T^{Max} - T_{kij}$, as the greatest value for τ_{kn} is T^{Max}. For the right hand side, this value should at least be T^{Max} when $x_{k'ijn'} = 0$. To always ensure this, we let $M = T^{Max} + T_{k'ij}$, and constraints (4.11) can now be written as follows:

$$\tau_{kn} - \left(T^{Max} - T_{kij}\right)\left(1 - x_{kijn}\right) - T_{kij} \leq \tau_{k'n'} + \left(T^{Max} + T_{k'ij}\right)\left(1 - x_{k'ijn'}\right) - T_{k'ij}$$
$$k \in \mathcal{K}^L, k' \in \mathcal{K}^S, n, n' \in \mathcal{N}, (i,j) \in \hat{\mathcal{A}}^L \big| (i,j) \in \hat{\mathcal{A}}^S$$

Improving the implementation of the model

It is fairly straightforward to identify some of the symmetries in the arc-flow formulation of the problem. For example, two vehicles of the same type may swap routes, yielding an equally good solution. In the project leading up to this thesis, an assessment of several symmetry-breaking constraints were conducted. However, pairwise sets of constraints were usually mutually exclusive, such that only one set can be implemented without removing unique parts of the solution space. Although all of the sets of symmetry-breaking constraints cut off significant parts of the solution space, several of the options led to longer computation times than if they were omitted from the implementation. The set that shortened the computation time the most, are given by constraints (4.16) and (4.17). These constraints ensure a lexicographic ordering of the plowing vehicles, such that for each of the two vehicle fleets, the vehicle with the lowest index number service more arcs than the one with the second lowest index number, and so on. With these constraints, the aforementioned symmetry only exist in the limit where two vehicles service the same number of arcs. These constraints are included in the implementation of the arc-flow model in the computational study.

$$\sum_{n \in \mathcal{N}} \sum_{(i,j) \in \hat{\mathcal{A}}^L} x_{kijn} \geq \sum_{n \in \mathcal{N}} \sum_{(i,j) \in \hat{\mathcal{A}}^L} x_{(k+1)ijn} \qquad k \in \mathcal{K}^L \big| k < |\mathcal{K}^L| \quad (4.16)$$

$$\sum_{n\in\mathcal{N}}\sum_{(i,j)\in\hat{\mathcal{A}}^S} x_{kijn} \geq \sum_{n\in\mathcal{N}}\sum_{(i,j)\in\hat{\mathcal{A}}^S} x_{(k+1)ijn} \qquad k \in \mathcal{K}^S \,|\, k < |\mathcal{K}^S| \quad (4.17)$$

Simplifications

In accordance with the problem description in Chapter 3, U-turns should be prohibited for the plowing trucks. Constraints (4.18) will prevent the trucks from going from node i to j and immediately return to node i. Note that prohibiting U-turns only makes sense for the larger trucks, as the smaller ones can easily cross roads and return to the node from where they came.

$$x_{kijn} + y_{kijn} + x_{kij(n+1)} + y_{kij(n+1)} \leq 1 \qquad k \in \mathcal{K}^L, (i,j) \in \mathcal{A}^L, n \in \mathcal{N}, \;|\; (j,i) \in \mathcal{A}^L, n < |\mathcal{N}| \quad (4.18)$$

Note that these constraints can only be formulated in this way if the road network of the underlying problem is actually traversable without having to perform U-turns. There are several known ways in which one can work its way around exceptions from this rule, such as adding more node-specific constraints or manipulating the graph. None of these methods are discussed further, as none of the instances used in our thesis require that the larger vehicles perform U-turns. One of the results obtained in our preliminary study of the MFARPOS was that the computation time was significant negatively affected when the U-turn constraints were added to the implementation. This is in contrast to actually shrinking the solution space. As the implementation of the arc-flow model only is meant as a tool for comparison to the column-generation approach presented in Chapter 5, and since disallowing U-turns is a different process in column generation, we relax the original problem and simply allow U-turns in both of these implementations.

With respect to left hand turns, the procedure for prohibiting these are complex, compared to that of U-turns. One way to solve it is to order pairs of consecutive arcs that form a left hand turn, and a new set of constraints that prohibit the vehicles from servicing (or even deadheading for a subset of the arc pairs) these in the order that involves a left hand turn. For problems related to the MFARPOS, using this approach to include turn restrictions has generally been disappointing with respect to computation

time, and are therefore often discouraged. An alternative, and much more popular way to include all sorts of turn restrictions, is to modify the graph in such a way that no extra constraints are required. In this case, all nodes that involve a potential forbidden left hand turn for the vehicles must be duplicated one or several times, where some arcs are removed from some nodes, and additional arcs are added. For reference, Clossey, Laporte, and Soriano (2001) describe this approach in detail. In this thesis, we simplify the problem to have no such turn penalties (U-turns are considered in some instances), due to the following. Firstly, the implementation would most likely aggravate the computation time for the MIP model, to which we compare the column generation model, and to improve the MIP implementation is beyond the scope of this thesis. Secondly, the road networks on which we test our models are fictitious, and no indication on whether a turn is a left turn exist. We therefore underline that the graphical representation of the instances used in this thesis can, but should not strictly, be viewed as a small scale geographical map of the road network they represent. We further elaborate on this in Chapter 7.

Problem complexity

The MFARPOS is $\mathcal{NP}$-hard, and thus there exists no known solution to solve the problem in deterministic polynomial time. To see this, consider the directed Rural Postman Problem (RPP), where a single fleet services a subset of the total number of arcs. This is equivalent of removing the plowing trucks, and to find the optimal routes for the smaller vehicles in the MFARPOS without the synchronization constraints. Lenstra and Kan (1976) prove that the RPP is $\mathcal{NP}$-hard, and since the MFARPOS is an extension of this, the MFARPOS is $\mathcal{NP}$-hard, as well.

Chapter 5

Dantzig-Wolfe Decomposition

As with many arc routing problems related to the MFARPOS, the computational power needed to solve a Branch-and-Bound algorithm based on their compact formulation to optimality, grows rapidly when arcs and vertices are added to the graph. Little research exists on decomposition of such problems, and we know of no such formulations for the problem studied in this thesis. In this chapter we therefore present a first-of-its-kind Dantzig-Wolfe decomposition of the mathematical model presented in Chapter 4. In Section 5.1 we describe a path-flow formulation of the problem, whereas Section 5.2 is concerned with the solution methodology, summarized in a pseudocode in Section 5.3. The chapter is concluded with a brief note on expansions in Section 5.4.

5.1 Path-flow formulation

In order to formulate the path-flow model, additional notation needs to be introduced. Let $\mathcal{R}_k$ be the set of all feasible routes for vehicle k, A_{kijr} is the number of times vehicle k plows arc (i, j) in route r, while T_{kr}^{Tot} is the total time of route r for vehicle k. For all the smaller vehicles, we assume that they enter the artificial end depot at time T^{Max}, which means that the smaller vehicle k driving route r leaves the artificial start depot at time $T^{Max} - T_{kr}^{Tot}$. To maintain the synchronization relations, the smaller vehicles may have to do some waiting in the route before they can service a given arc (given that they leave the start depot at time 0). Without degenerating the solution, all necessary waiting can just as well be deployed before the vehicle starts its route. Let the variables w_k state how much earlier vehicle $k \in \mathcal{K}^S$ should leave the artificial start depot when driving a given route, such that all the waiting time, due to the synchronization relation, is done

in the start depot. As all the lanes on a given segment have to be serviced before the corresponding sidewalk, let B_{kijr} state the starting time of the last plow job for vehicle $k \in \mathcal{K}^L$ on arc (i, j) in route r, and the starting time for the first plow job for vehicle $k \in \mathcal{K}^S$. Due to the formulation of the constraints including B_{kijr}; if arc (i, j) is not serviced on route r, the value of B_{kijr} is 0 if $k \in \mathcal{K}^L$, and T^{Max} if $k \in \mathcal{K}^S$. Further, let the variables λ_{kr} indicate whether vehicle k is assigned to route r or not. Let the variables t^P_{ij} represent the earliest starting time of service on arc (i, j) for a vehicle $k \in \mathcal{K}^S$ if there is a synchronization requirement associated with the arc.

With this notation, the problem can be formulated as follows:

Objective function

$$\text{Minimize} \quad t^{MS} \tag{5.1}$$

s.t.

$$\sum_{k\in\mathcal{K}^L}\sum_{r\in\mathcal{R}_k} A_{kijr}\lambda_{kr} \geq R^L_{ij} \qquad (i,j) \in \hat{\mathcal{A}}^L \tag{5.2}$$

$$\sum_{k\in\mathcal{K}^S}\sum_{r\in\mathcal{R}_k} A_{kijr}\lambda_{kr} \geq R^S_{ij} \qquad (i,j) \in \hat{\mathcal{A}}^S \tag{5.3}$$

$$\sum_{r\in\mathcal{R}_k} \lambda_{kr} = 1 \qquad k \in \mathcal{K}^L \tag{5.4}$$

$$\sum_{r\in\mathcal{R}_k} \lambda_{kr} = 1 \qquad k \in \mathcal{K}^S \tag{5.5}$$

$$\sum_{r\in\mathcal{R}_k} B_{kijr}\lambda_{kr} \leq t^P_{ij} \qquad k \in \mathcal{K}^L, (i,j) \in \hat{\mathcal{A}}^L \big| (i,j) \in \hat{\mathcal{A}}^S \tag{5.6}$$

$$t^P_{ij} - \sum_{r\in\mathcal{R}_k} B_{kijr}\lambda_{kr} + w_k \leq 0 \qquad k \in \mathcal{K}^S, (i,j) \in \hat{\mathcal{A}}^S \big| (i,j) \in \hat{\mathcal{A}}^L \tag{5.7}$$

$$\sum_{r\in\mathcal{R}_k} T^{Tot}_{kr}\lambda_{kr} + w_k \leq T^{Max} \qquad k \in \mathcal{K}^S \tag{5.8}$$

$$\sum_{r\in\mathcal{R}_k} T^{Tot}_{kr}\lambda_{kr} \leq t^{MS} \qquad k \in \mathcal{K}^L \tag{5.9}$$

$$T^{Max} - w_k \leq t^{MS} \qquad k \in \mathcal{K}^S \tag{5.10}$$

$$\lambda_{kr} \in \{0,1\} \qquad k \in \mathcal{K}, r \in \mathcal{R}_k \tag{5.11}$$

$$t^P_{ij} \geq 0 \qquad (i,j) \in \hat{\mathcal{A}}^L \big| (i,j) \in \hat{\mathcal{A}}^S \tag{5.12}$$

$$w_k \geq 0 \qquad k \in \mathcal{K}^S \tag{5.13}$$

The objective function (5.1) aims to minimize the makespan of the schedule. Constraints (5.2) and (5.3) state that all arcs with service demand shall be serviced, while equations (5.4) and (5.5) limit each vehicle to drive exactly one route. Constraints (5.6) and (5.7) ensure that the synchronization relations between the lanes and sidewalks hold. Constraints (5.8) set an upper bound on the waiting time variables, while constraints (5.9) and (5.10) track the makespan of the schedule. Finally, constraints (5.11) impose binary restrictions on the routes, and constraints (5.12) and (5.13) ensure that the synchronization time variables and waiting time variables are non-negative.

5.2 Solution methodology

Each λ-variable, henceforth referred to as *column* in the path-flow formulation, is described by a path through the graph $\mathcal{G}$. The number of feasible paths through the graph grows exponentially with the number of nodes and arcs. Therefore, for large problem instances, it will be impractical or even impossible, to generate all columns for such a formulation. To circumvent this problem, we propose solving the path-flow formulation using a Branch-and-Price approach, where only a subset of all columns are explicitly considered, while implicitly accounting for the remaining columns.

The Branch-and-Price method implicitly enumerates the set of possible solutions to the problem, by using a Branch-and-Bound approach. The problem is reformulated into two types of problems, which are considered in each node in the B&B tree: the restricted master problem (RMP), which is the linear relaxation of the path-flow formulation, only including a subset of all the possible columns; and one pricing problem per vehicle (these are generally called the subproblems - SPs). Each of the SPs uses the dual solution of the RMP to generate new columns with negative reduced cost (for a minimization problem), which is passed on to the RMP. The RMP is solved again with the new columns, where the new dual solution then is used to re-solve the SPs. These iterations between the RMP and the SPs continue until all additional columns generated by the SPs cease to have a negative reduced cost. At this point, no more columns can improve the RMP, and the optimal solution to the RMP in the node is obtained. At this point, three possible outcomes must be considered: 1) the value of the optimal solution in the node is higher (in a minimization problem) than the best known integer solution to the problem - at which point no further branching is needed; 2) the value of the solution is better than the best known integer solution and the solution itself is an integer solution - at which point the solution is noted as the best known integer solution and no

further branching is required; and 3) the value of the solution in the node is better than the best known integer solution, but it violates some of the integral requirements of the original path-flow formulation - at which point the solution is noted as the new lower bound in the node and further branching is done by creating two new nodes, based on some binary properties of the problem. Branching is performed until all possible nodes have either been branched on, cut off, or contain an integer solution. The best integer solution is then returned as the optimal solution to the problem.

5.2.1 Pricing of columns

Let α^L_{ij} and α^S_{ij} be the dual values of constraints (5.2) and (5.3) from a given iteration of the column generation algorithm, and β^L_k and β^S_k the dual values of constraints (5.4) and (5.5). Further, let γ^L_{kij} and γ^S_{kij} be the dual values of constraints (5.6) and (5.7), and σ^S_k and σ^L_K the dual values of constraints (5.8) and (5.9). The columns of the path-flow formulation are priced according to equation (5.14) for the larger trucks, and according to equation (5.15) for the smaller vehicles.

$$\overline{c}^L_{kr} = -\beta^L_k - \sum_{(i,j)\in\hat{\mathcal{A}}^L} A_{kijr}\alpha^L_{ij} - \sum_{(i,j)\in\hat{\mathcal{A}}^L} B_{kijr}\gamma^L_{kij} - T^{Tot}_{kr}\sigma^L_k \tag{5.14}$$

$$\overline{c}^S_{kr} = -\beta^S_k - \sum_{(i,j)\in\hat{\mathcal{A}}^S} A_{kijr}\alpha^S_{ij} + \sum_{(i,j)\in\hat{\mathcal{A}}^S} B_{kijr}\gamma^S_{kij} - T^{Tot}_{kr}\sigma^S_k \tag{5.15}$$

5.2.2 The subproblems

Two different sets of subproblems are needed - one for each type of vehicle - in the aforementioned formulation. These subproblems are solved for each of the vehicles in the corresponding fleet. The two unique subproblems are presented below, both have the same set of constraints, while their objective function differs. Let t^{tot}_k be the total time vehicle k uses on its route, while t^L_{ij} tracks the last time arc $(i,j) \in \hat{\mathcal{A}}^L$ is serviced, and t^S_{ij} tracks the first time arc $(i,j) \in \hat{\mathcal{A}}^S$ is serviced. The subproblem for a vehicle k may be formulated in the following way:

Objective functions

If $k \in \mathcal{K}^L$

$$\text{Minimize} - \beta_k^L - \sum_{n \in \mathcal{N}} \sum_{(i,j) \in \hat{\mathcal{A}}^L} x_{kijn} \alpha_{ij}^L - \sum_{(i,j) \in \hat{\mathcal{A}}^L} t_{ij}^L \gamma_{kij}^L - t_k^{tot} \sigma_k^L \tag{5.16}$$

If $k \in \mathcal{K}^S$

$$\text{Minimize} - \beta_k^S - \sum_{n \in \mathcal{N}} \sum_{(i,j) \in \hat{\mathcal{A}}^S} x_{kijn} \alpha_{ij}^S + \sum_{(i,j) \in \hat{\mathcal{A}}^S} t_{ij}^S \gamma_{kij}^S - t_k^{tot} \sigma_k^S \tag{5.17}$$

s.t.

$$\sum_{j \in \delta_k^+(\hat{O})} y_{k\hat{O}j1} = 1 \tag{5.18}$$

$$\sum_{n \in \mathcal{N}} \sum_{j \in \delta_k^-(\hat{D})} y_{kj\hat{D}n} = 1 \tag{5.19}$$

$$\sum_{j \in \delta_k^+(\hat{O})} \left(x_{k\hat{O}jn} + y_{k\hat{O}jn} \right) = 0 \qquad n \in \mathcal{N} \big| n > 1 \tag{5.20}$$

$$\sum_{i \in \delta_k^-(j)} \left(x_{kijn} + y_{kijn} \right) - \sum_{i \in \delta_k^+(j)} \left(x_{kji(n+1)} + y_{kji(n+1)} \right) = 0 \qquad j \in \mathcal{V}, \quad n \in \mathcal{N} \big| n < \overline{n} \tag{5.21}$$

$$\sum_{n \in \mathcal{N}} x_{kijn} \leq R_{ij}^L \qquad (i,j) \in \hat{\mathcal{A}}^L \tag{5.22}$$

$$\sum_{(i,j) \in \mathcal{A}_k} \left(x_{kijn} + y_{kijn} \right) \leq 1 \qquad n \in \mathcal{N}, n > 1 \tag{5.23}$$

$$\tau_{k(n-1)} + \sum_{(i,j) \in \mathcal{A}_k} \left(T_{kij} x_{kijn} + T_{kij}^D y_{kijn} \right) \leq \tau_{kn} \qquad n \in \mathcal{N} | n > 1 \tag{5.24}$$

$$\tau_{kn} - T^{Max} \left(1 - x_{kijn} \right) \leq t_{ij}^L \qquad (i,j) \in \hat{\mathcal{A}}^L, \quad n \in \mathcal{N} \tag{5.25}$$

$$\tau_{kn} \leq t_k^{tot} \qquad n \in \mathcal{N} \tag{5.26}$$

$$x_{kijn} \in \{0,1\} \qquad (i,j) \in \mathcal{A}_k, \quad n \in \mathcal{N} \tag{5.27}$$

$$y_{kijn} \in \{0,1\} \qquad (i,j) \in \mathcal{A}_k, \quad n \in \mathcal{N} \tag{5.28}$$

$$\tau_{kn} \geq 0 \qquad n \in \mathcal{N} \tag{5.29}$$

$$t_{ij}^L \geq 0 \qquad (i,j) \in \hat{\mathcal{A}}^L \tag{5.30}$$

The objective of the subproblems is to find the minimum priced route for a given vehicle. Constraints (5.18) through (5.23) conserve the flow through the network from the start depot to the end depot, while constraints (5.24) through (5.26) keep track of the time. Finally, the remaining constraints force the decision variables x_{kijn} and y_{kijn} to be binary, and τ_{kn} and t^L_{ij} to be non-negative.

The constraints for the subproblem presented above are related to any vehicle $k \in \mathcal{K}^L$. For a vehicle $k \in \mathcal{K}^S$ the constraints are equivalent, except for some minor changes in constraints (5.22), (5.25) and (5.30). In constraints (5.22), R^S_{ij} and $(i,j) \in \hat{\mathcal{A}}^S$ replace R^L_{ij} and $(i,j) \in \hat{\mathcal{A}}^L$, respectively. In constraints (5.30), t^S_{ij} and $(i,j) \in \hat{\mathcal{A}}^S$ replace t^L_{ij} and $(i,j) \in \hat{\mathcal{A}}^L$. In order to track the first time an arc is serviced for the smaller vehicles, constraints (5.31) replace (5.25) in the subproblem for a vehicle $k \in \mathcal{K}^S$.

$$\tau_{kn} - T^{Max}\left(1 - x_{kijn} + \sum_{n'=1}^{n-1} x_{kijn'}\right) \leq t^S_{ij} \qquad (i,j) \in \hat{\mathcal{A}}^S, \quad n \in \mathcal{N} \tag{5.31}$$

5.2.3 A labeling algorithm for the subproblems

The subproblems presented in Section 5.2.2 are variations of the Shortest Path Problem with Resource Constraints (SPPRC), as presented by Irnich and Desaulniers (2005). In order to solve these, we use a modified version of the Generic Dynamic Programming (GDP) algorithm suggested by the same authors. As a result of the different pricing of columns for the two subproblems, a specific labeling algorithm for each of the subproblems has been developed.

The subproblem for the plowing trucks is solved according to the pseudocode provided in Algorithm 1. Let U be the set of unprocessed labels (initially containing the artificial starting depot $\hat{O}$), while $\mathcal{U}_i$ and $\mathcal{P}_i$ are the sets of unprocessed and processed labels in node i, respectively. If there are any labels left in U, and the stop criteria is not met, the function removes the label with the lowest reduced cost. This label is extended (as described in Section 5.2.4) along all feasible arcs. If the new label, L', is not dominated (according to the description in Section 5.2.5), it is added to the unprocessed sets. Should the extended label be in the artificial end depot, $\hat{D}$, with a negative reduced cost, it is added to the list of labels for the RMP ($listOfLabels$). If the number of iterations since the function last found a new label for the RMP is greater than $tresholdCounter$, or the size of the list of labels for RMP is greater than $thresholdList$, the list of labels is

returned to the RMP. The algorithm returns $null$ if no columns with negative reduced cost exists when the set U is empty.

Algorithm 1 differs from the GDP in the way that it does not guarantee to return the path with the most negative reduced cost. As the only thing restricting the resource extension function is the maximum time of a route, the number of possible extensions in the algorithm is generally large. Therefore, in order to not try all feasible extensions every time the subproblem is solved, an iteration counter is included to store the number of iterations since a column with negative reduced cost was generated. The algorithm also includes a parameter indicating the maximum number of columns that potentially could be sent to the RMP. Preliminary testing shows that including these parameters, almost no columns that improve the solution are omitted, while the computation time in every subproblem is significantly reduced.

With respect to the smaller vehicles, constraints (5.7) in the RMP yield a negative value to the γ^S_{kij}-term in the pricing of columns for the smaller vehicles in equations (5.15). Hence, we would like the values of B_{kijr} to be as large as possible, which indicate that the arcs should be serviced close to the time limit of a route. At the same time, we would like to minimize the makespan. Therefore, we find the shortest path for the smaller vehicles by traverse the graph in reversed order. Accordingly, the labeling algorithm for the subproblem of the smaller vehicles starts in the end depot at time T^{Max}, and traverses the graph backwards, while subtracting the time used. By doing this, the values of B_{kijr} are set as large as possible, and at the same time making it possible to utilize the main features of the GDP algorithm. Briefly, this labeling algorithm only differ from Algorithm 1 in the initialization phase, where $U = \{L(\hat{D})\}$, and in the if-statement that tests whether the column has a negative reduced cost. Specifically, it's checking if $node(L') = \hat{O}$ for the smaller vehicles.

Algorithm 1 Labeling algorithm for the plowing trucks

```
1:  U = {L(Ô)}
2:  while U ≠ ∅ do
3:      sort U on the lowest reduced cost
4:      L = removeFirst(U)
5:      for each feasible extension of L → L' do
6:          if node(L') = D̂ and cost(L') < 0 then
7:              add L' to listOfLabels
8:              counter = 0
9:          end if
10:         i = node(L')
```

```
        if no label in $\mathcal{U}_i$ and $\mathcal{P}_i$ dominates L' then
            remove all labels in $\mathcal{U}_i$ and U that are dominated by L'
            $\mathcal{U}_i = \mathcal{U}_i \cup$ {L'}
            U = U ∪ L'
        end if
    end for
    j = node(L)
    $\mathcal{P}_j = \mathcal{P}_j \cup L$
    if listOfLabels ≠ ∅ and counter > $tresholdCounter$ then
        return listOfLabels
    end if
    if listOfLabels.length > $thresholdList$ then
        return listOfLabels
    end if
    counter = counter + 1
end while
if listOfLabels ≠ ∅ then
    return listOfLabels
end if
return null
```

5.2.4 Labels and label extension

The data stored in each label are presented in Table 5.1. We note that for the remainder of this thesis, the notation $\eta(L)$ is used to refer to the node of label L, and similar notation is used for the rest of the label data.

Table 5.1: The data stored in each label.

Data	Description
η	The node of the label
ϕ	The predecessing label
t	The arriving time at the node
c	The accumulated reduced cost
θ	Binary variable indicating whether the vehicle serviced ($\theta = 1$) or deadheaded ($\theta = 0$) the last traversed arc
B_{ij}	The starting time of the last plow job of arc (i, j) for the plowing trucks, and the starting time of the first plow job of arc (i, j) for the smaller vehicles
A_{ij}	The number of times arc (i, j) is serviced

For the plowing trucks, if a label L is extended along an arc $(\eta(L), j)$, two new labels L' are created at node j: one for each value of θ, since the vehicle can either service

($\theta = 1$) or deadhead ($\theta = 0$) the arc from label L to label L'. Let T_{ki}^{Depot} be the shortest time from node i to the destination depot. The label data are updated as follows:

$$\eta(L') = j \tag{5.32}$$

$$\phi(L') = L \tag{5.33}$$

$$t(L') = \begin{cases} t(L) + T_{k\eta(L)j} & \text{if} \quad \theta = 1 \\ t(L) + T^{D}_{k\eta(L)j} & \text{if} \quad \theta = 0 \end{cases} \tag{5.34}$$

$$c(L') = c(L) - \left[t(L') - t(L)\right]\sigma_k^L + \begin{cases} -\beta_k^L & \text{if} \quad \eta(L) = \hat{O} \\ -\alpha^L_{\eta(L)j} - \left[t(L) - B_{k\eta(L)j}(L)\right]\gamma_{k\eta(L)j} & \text{if} \quad \theta = 1 \\ 0 & \text{otherwise} \end{cases} \tag{5.35}$$

For $\forall (i,j) \in \hat{\mathcal{A}}^L$

$$B_{kij}(L') = \begin{cases} t(L) & \text{if} \quad (i,j) = (\eta(L), \eta(L')) \ \wedge \ \theta = 1 \\ B_{kij}(L) & \text{otherwise} \end{cases} \tag{5.36}$$

$$A_{kij}(L') = A_{kij}(L) + \theta \tag{5.37}$$

Equations (5.32) through (5.35) update the current node, the predecessing label, the time spent, and the cost component of the label, respectively. Equations (5.36) and (5.37) update the value of the starting time of the arcs' last service, in addition to the total number of times they are serviced.

For $\theta = 1$ an extension is feasible if:

$$A_{\eta(L)j} + 1 \leq R^L_{\eta(L)j} \tag{5.38}$$

$$t(L) + T_{\eta(L)j} + T_{kj}^{Depot} \leq T^{Max} \tag{5.39}$$

For $\theta = 0$ an extension is feasible if:

$$t(L) + T^{D}_{\eta(L)j} + T^{Depot}_{kj} \leq T^{Max} \tag{5.40}$$

For the smaller vehicles, the graph is traversed in reverse order. If a label L is extended along an arc $(j, \eta(L))$, two new labels L' are created at node j, one for each value of θ. Let T^{Depot}_{ki} be the shortest time from the starting depot to node i. The label data is updated as follows:

$$\eta(L') = j \tag{5.41}$$

$$\phi(L') = L \tag{5.42}$$

$$t(L') = \begin{cases} t(L) - T_{kj\eta(L)} & \text{if} \quad \theta = 1 \\ t(L) - T^{D}_{kj\eta(L)} & \text{if} \quad \theta = 0 \end{cases} \tag{5.43}$$

$$c(L') = c(L) - \left[t(L) - t(L')\right]\sigma^{S}_{k} + \begin{cases} -\beta^{S}_{k} & \text{if} \quad \eta(L) = \hat{D} \\ -\alpha^{S}_{j\eta(L)} - \gamma^{S}_{kj\eta(L)}\, t(L') & \text{if} \quad \theta = 1 \wedge A_{kij}(L) = 0 \\ -\alpha^{S}_{j\eta(L)} - \gamma^{S}_{kj\eta(L)}\left[t(L') - B_{kj\eta(L)}(L)\right] & \text{if} \quad \theta = 1 \wedge A_{kij}(L) > 0 \\ 0 & \text{otherwise} \end{cases} \tag{5.44}$$

For $\forall (i,j) \in \hat{\mathcal{A}}^{S}$

$$B_{kij}(L') = \begin{cases} t(L') & \text{if} \quad (i,j) = (\eta(L'), \eta(L)) \ \wedge \ \theta = 1 \\ B_{kij}(L) & \text{otherwise} \end{cases} \tag{5.45}$$

$$A_{kij}(L') = A_{kij}(L) + \theta \tag{5.46}$$

Equations (5.41) through (5.44) update the current node, the predecessing label, the time spent, and the cost component of the label, respectively. Equations (5.45) and

(5.46) update the value of the starting time of the arcs' last service, in addition to the total number of times they are serviced.

For $\theta = 1$, an extension is feasible if:

$$A_{j\eta(L)} + 1 \leq R^S_{j\eta(L)} \tag{5.47}$$

$$t(L) - T_{j\eta(L)} - T^{Depot}_{kj} \geq 0 \tag{5.48}$$

For $\theta = 0$, an extension is feasible if:

$$t(L) - T^D_{j\eta(L)} - T^{Depot}_{kj} \geq 0 \tag{5.49}$$

5.2.5 Dominance criteria

The dominance criteria used to remove dominated labels for the plowing trucks are given in Proposition 1. For simplicity let the set $\mathcal{A}_\theta = \{(i,j) \mid A_{ij}(L_1) > A_{ij}(L_2)\}$, and $\underline{T}_{kij}$ be the shortest time from node i to j for vehicle k.

Proposition 1. *A label L_1 dominates L_2 if:*

1. $\eta(L_1) = \eta(L_2)$
2. $t(L_1) \leq t(L_2)$
3. $c(L_1) \leq c(L_2) + \sum_{(i,j)\in\mathcal{A}_\theta} min\left\{\left(-\alpha^L_{ij}\left[A_{ij}(L_1) - A_{ij}(L_2)\right] - \left[t(L_2) + \underline{T}_{k\eta(L_2)i}\right]\gamma^L_{kij}\right), 0\right\}$

Proof. Let p be a feasible path extending L_2 from $\eta(L_2)$ $\left(=\eta(L_1)\right)$ to a given node i. From path p we define $\mathcal{A}_p$ as the set of the arcs in the path, and $\vec{\theta}_p = [\theta_1, ..., \theta_{|\vec{\theta}|}]$ as a vector with the sequence of the traversed arcs, and whether the arc at position m is serviced ($\theta_m = 1$), or not ($\theta_m = 0$). For a compact notation, we use the Kronecker-symbol with $\delta_{jlm} = 1$ if arc (j,l) is at position m in the vector, and $\delta_{jlm} = 0$ otherwise. And for simplicity we name $p_1 = (L_1, p)$ as the concatenation of the partial path represented by L_1 and p, and likewise $p_2 = (L_2, p)$. Let L_{p_1} and L_{p_2} be the labels in node i where their predecessors have followed the paths p_1 and p_2, respectively.

First, in events where $\forall (j,l) \in \mathcal{A}_p : A_{jl}(L_1) \leq A_{jl}(L_2)$, every feasible extension of L_2 fulfill the inequalities (5.50) and (5.51).

$$A_{jl}(L_2) + \sum_{m=1}^{|\vec{\theta}_p|} \delta_{jlm}\theta_m \leq R_{jl} \qquad \forall (j,l) \in \mathcal{A}_p \quad (5.50)$$

$$t(L_2) + \sum_{(j,l)\in\mathcal{A}_p} \sum_{m=1}^{|\vec{\theta}_p|} \delta_{jlm}\left[T_{jlk}\theta_m + T^D_{jlk}(1-\theta_m)\right] + T_i^{Depot} \leq T^{Max} \quad (5.51)$$

As $\forall (j,l) \in \mathcal{A}_p : A_{jl}(L_1) \leq A_{jl}(L_2)$ and (if criterion 2 holds) $t(L_1) \leq t(L_2)$, if inequalities (5.50) and (5.51) hold, this implies that inequalities (5.52) and (5.53) also hold. And thereby, every feasible extension of L_2 is feasible for L_1, as well.

$$A_{jl}(L_1) + \sum_{m=1}^{|\vec{\theta}_p|} \delta_{jlm}\theta_m \leq R_{jl} \qquad \forall (j,l) \in \mathcal{A}_p \quad (5.52)$$

$$t(L_1) + \sum_{(j,l)\in\mathcal{A}_p} \sum_{m=1}^{|\vec{\theta}_p|} \delta_{jlm}\left[T_{jlk}\theta_m + T^D_{jlk}(1-\theta_m)\right] + T_i^{Depot} \leq T^{Max} \quad (5.53)$$

The reduced costs of L_{p_1} and L_{p_2}, $c(L_{p_1})$ and $c(L_{p_2})$, can be expressed by the equations (5.54) and (5.55), respectively.

$$\begin{aligned} c(L_{p_1}) = c(L_1) &- \sum_{(j,l)\in A_p} \sum_{m=1}^{|\vec{\theta}_p|} \alpha^L_{jl}\delta_{jlm}\theta_m \\ &- \sigma^L_k \left(t(L_1) + \sum_{(j,l)\in A_p} \sum_{m=1}^{|\vec{\theta}_p|} \delta_{jlm}\left(T_{kjl}\theta_m + T^D_{kjl}[1-\theta_m]\right)\right) \\ &- \sum_{(j,l)\in A_p} \sum_{m=1}^{|\vec{\theta}_p|} \left(t(L_1) + \sum_{\substack{(q,r)= \\ (\eta(L_1),s), \\ \ldots,(t,j)}} \sum_{m'=1}^{m} \delta_{qrm'}\left(T_{kqr}\theta_{m'} + T^D_{kqr}[1-\theta_{m'}]\right)\right) \delta_{jlm}\theta_m\gamma^L_{kjl} \end{aligned} \quad (5.54)$$

$$
\begin{aligned}
c(L_{p_2}) = c(L_2) &- \sum_{(j,l)\in A_p} \sum_{m=1}^{|\vec{\theta}_p|} \alpha^L_{jl}\delta_{jlm}\theta_m \\
&- \sigma^L_k \left(t(L_2) + \sum_{(j,l)\in A_p} \sum_{m=1}^{|\vec{\theta}_p|} \delta_{jlm}\left(T_{kjl}\theta_m + T^D_{kjl}[1-\theta_m] \right) \right) \\
&- \sum_{(j,l)\in A_p} \sum_{m=1}^{|\vec{\theta}_p|} \left(t(L_2) + \sum_{\substack{(q,r)= \\ (\eta(L_2),s), \\ \ldots,(t,j)}} \sum_{m'=1}^{m} \delta_{qrm'}\left(T_{kqr}\theta_{m'} + T^D_{kqr}[1-\theta_{m'}] \right) \right) \delta_{jlm}\theta_m\gamma^L_{kjl}
\end{aligned}
\tag{5.55}
$$

By a pairwise comparison, the following can be obtained. For the first term, criterion 3 simply requires that $c(L_1) \leq c(L_2)$, since $\mathcal{A}_\theta = \emptyset$. As $\vec{\theta}_p$ is equal for both labels, the second term is also equal. As criterion 2 demand that $t(L_1) \leq t(L_2)$, and since the labels follow the same route, everything inside the brackets in the third and fourth term is equal or greater for L_{p_2}. As both $\gamma^L_{kjl} \leq 0$ and $\sigma^L_k \leq 0$, the third and fourth term will yield a greater positive value for L_{p_2} than L_{p_1}. Thereby, if criteria 1 through 3 is met, for every feasible extension p of L_2 one can conclude that

$$\forall (j,l) \in \mathcal{A}_p : A_{jl}(L_1) \leq A_{jl}(L_2)$$

$$\implies c(L_{p_1}) \leq c(L_{p_2})$$

By looking at a case where $\exists (i,j) \in \mathcal{A}_p : A_{ij}(L_1) > A_{ij}(L_2)$, for every feasible extended path p of L_2 all $\mathcal{A}_p$ are feasible for L_1, but not all $\vec{\theta}_p$ are feasible, since p_1 would exceed the number of times an arc can be serviced, R^L_{ij}, for some $\vec{\theta}_p$. But there exist other vectors that make the extension feasible for L_1, and one of those is where $\forall (j,l) \in \mathcal{A}_\theta : \theta_m = 0 \mid \delta_{jlm} = 1$. We denote this vector as $\vec{\theta}^1_p$, where θ^1_m is the arc at position m in the vector. Given that inequalities (5.56) and (5.57) hold, this implies that inequality (5.58) holds, and we see that extension p with $\vec{\theta}^1_p$ is feasible for L_1.

$$t(L_2) + \sum_{(j,l)\in\mathcal{A}_p} \sum_{m=1}^{|\vec{\theta}_p|} \delta_{jlm}(T_{jlk}\theta_m + T^D_{jlk}(1-\theta_m)) + T^{Depot}_{ki} \leq T^{Max} \tag{5.56}$$

$$T^D_{kjl} \leq T_{kjl} \qquad \forall (j,l) \in \hat{\mathcal{A}}^L \tag{5.57}$$

$$t(L_1) + \sum_{(j,l)\in\mathcal{A}_p} \sum_{m=1}^{|\tilde{\theta}_p^1|} \delta_{jlm}(T_{jlk}\theta_m^1 + T_{jlk}^D(1-\theta_m^1)) + T_{ki}^{Depot} \leq T^{Max} \tag{5.58}$$

However, since $\alpha_{jl}^L \geq 0$, this means that $c(L_{p_2})$ can add a negative value which $c(L_{p_1})$ cannot. Considering this, criterion 3 has the term

$$\sum_{(j,l)\in\mathcal{A}_\theta} min\left\{\left(-\alpha_{jl}^L\left[A_{jl}(L_1) - A_{jl}(L_2)\right] - \left[t(L_2) + \underline{T}_{k\eta(L_2)j}\right]\gamma_{kjl}^L\right), 0\right\}$$

This term iterates through all of the arcs that L_1 has plowed more times than L_2, and if

$$-\alpha_{jl}^L\left[A_{jl}(L_1) - A_{jl}(L_2)\right] < \left[t(L_2) + \underline{T}_{k\eta(L_2)j}\right]\gamma_{kjl}^L$$

this means that L_{p_2} could have added a negative value to its reduced cost by servicing the arc as soon as possible, which is at time $t(L_2) + \underline{T}_{k\eta(L_2)j}$. If L_{p_2} cannot get a lower reduced cost than L_{p_1} by servicing all of these arcs, then

$$c(L_1) \leq c(L_2) + \sum_{(i,j)\in\mathcal{A}_\theta} min\left\{\left(-\alpha_{ij}^L\left[A_{ij}(L_1) - A_{ij}(L_2)\right] - \left[t(L_2) + \underline{T}_{k\eta(L_2)j}\right]\gamma_{kij}^L\right), 0\right\}$$

for all feasible extensions of L_2. As the calculation above assume that the time at which an arc is serviced is as early as possible, the time at which it is serviced in the path may be later. That is, in some cases, γ_{kjl}^L is multiplied with a number too small, but as $\gamma_{kjl}^L \leq 0$, this also implies that the third dominance criterion is not equally strong in all situations.

Reviewing the expression for $c(L_{p_1})$ and $c(L_{p_2})$ from equations (5.54) and (5.55), respectively, we see that in the first term $c(L_1) \leq c(L_2)$ (from criterion 3), since

$$\sum_{(j,l)\in\mathcal{A}_\theta} min\left\{\left(-\alpha_{jl}^L\left(A_{jl}(L_1) - A_{jl}(L_2)\right) - \left(t(L_2) + \underline{T}_{k\eta(L_2)j}\right)\gamma_{kjl}^L\right), 0\right\} \leq 0$$

If p_2 service some arcs that is left unserved in p_1, the value in the second and fourth term could differ in the two formulations. However, this is considered in the last term in criterion 3, as described above. As criterion 2 holds, and $\forall(j,l) \in \hat{\mathcal{A}}^L : T_{kjl}^D \leq T_{kjl}$, it implies that the number inside the bracket in the third term cannot have a value smaller for L_{p_2} than it does for L_{p_1}. Since $\sigma_k^L \leq 0$, the third term adds a greater value for L_2 than L_1. Therefore, given that the three criteria is met, $c(L_{p_1}) \leq c(L_{p_2})$, and we can conclude that label L_1 dominates label L_2.

□

The subproblem for the smaller vehicles has the dominance criterion as given in Proposition 2. For simplicity, let the set $\mathcal{A}_\theta = \{(i,j) \mid A_{ij}(L_1) > A_{ij}(L_2)\}$, and $\underline{T}_{kij}$ be the shortest time from node i to j for vehicle k.

Proposition 2. *A label L_1 dominates L_2 if:*

1. $\eta(L_1) = \eta(L_2)$
2. $t(L_1) \geq t(L_2)$
3. $c(L_1) \leq c(L_2) + \sum_{(i,j)\in\mathcal{A}_\theta} \left(-\alpha^S_{ij}\left[A_{ij}(L_1) - A_{ij}(L_2)\right] + \gamma^S_{kij}\left[t(L_2) - \underline{T}_{kj\eta(L_2)} - T^D_{kij}\right]\right)$

The proof for Proposition 2 can be derived in a similar fashion as the proof for Proposition 1.

5.2.6 Acceleration strategies

In order to shorten the computation time and get a better root node solution, the genetic algorithm presented in Chapter 6 can be used to generate a set of columns that are used when the RMP is initially solved. This is often referred to as a *warm start* of the column generation. Depending on the quality of the solutions provided by the heuristic, we expect the number of iterations between the RMP and the subproblems to be decreased significantly - which can reduce the total computation time. The genetic algorithm always provides a feasible solution, whose objective value is set as the upper bound, T^{Max}. Tightening this maximum time parameter makes fewer label extensions feasible in the subproblems, and the label setting algorithm can exclude more columns. Note that not only the columns from the best heuristic solution is added to the RMP in the initial phase. The genetic algorithm has a pool of solutions (a population), where each solution consists of one column per vehicle. Although all but the best solution may have a higher makespan value than the best found solution, there is nothing prohibiting single columns in the poorer solutions to have a value on par with or better than the best makespan of an entire solution. Therefore, all columns with a value less than or equal to the best solution are also added to the RMP. Should the B&P algorithm find an integer solution during the search that is better than the best known IP solution, the T^{Max} value is updated to be equal that solution, which reduces the number of feasible labels even more.

5.2.7 Branching

For non-integer solutions in the root node, a branching strategy in the B&B tree has to be devised in order to obtain an integer solution. In this thesis, we consider two branching strategies used in a hierarchical fashion. The first strategy branches on whether a given vehicle services a given arc (given that the arc only needs to be serviced once). The 0-branch is imposed by removing all paths created in the subproblem for the given vehicle where the given arc is serviced. The 1-branch is imposed by discarding all paths for the given vehicle that does not service the given arc, and also all paths for the other vehicles of the same type that do service the arc. For the 1-branch, the dominance criteria must be modified. A path that includes servicing of the respective arc can dominate all the other paths, while a path that leaves the arc unserviced, can only dominate other paths where the branched arc is not serviced.

The second branching strategy is to branch on whether a given arc in a vehicle's plowing sequence is serviced immediately after another given arc. The 0-branch is imposed by removing all paths where the two arcs are consecutive in the vehicle's plowing sequence. The 1-branch is imposed by removing all the paths for the given vehicle where the arcs are not subsequent in the plowing sequence. The dominance criteria for the 1-branch in this branching strategy are modified as well. A path that includes servicing the subsequent arcs can dominate all the other paths, while a path in which they are not serviced can only dominate other paths where the arcs are not being serviced.

For both branching strategies, the nodes in the B&B tree are processed in a *best first* order, depending on the lower bound of their parent node. The selection among equally good nodes is done by *depth first* sequence.

5.3 Pseudocode for the Branch-and-Price algorithm

The pseudocode for the implementation of the B&P model, as described throughout this chapter, is given in Algorithm 2. The algorithm is fairly similar to other B&P implementations, starting by initializing the problem with a warm start (line 1 through line 5), and then iterates between the RMP (line 18) and the SPs (line 20 through line 25) through the B&B tree (line 6 through line 51). In line 39 through line 43 the algorithm branches on the strategies as described in Section 5.2.7. Our implementation is special with respect to the general B&P scheme, as it checks whether a better integer solution is found, such that the labeling algorithm can be modified. In the pseudocode, this is ensured by line 34 and line 47. If a better integer solution has been found, the value of

T^{Max} is updated. As described in 5.2.6, this reduction affects the possible length of the columns generated in the SPs, and was therefore included as an acceleration strategy.

Since preliminary runs of the algorithm showed that a lot of branching were needed in order to get an integer solution, line 44 through 50 were added to the procedure. This part of the code solves the RMP as a MIP with the set of all columns generated. As this is a time consuming process, the MIP does not run in every node, but rather in fixed intervals between nodes. This interval is dependent on the size of the test instance.

Algorithm 2 Branch-and-Price

```
1: Initialize problem, adding root node to B&B tree
2: Solve heuristic
3: T^Max = heuristic solution
4: Include paths from heuristic that are ≤ T^Max
5: optimalSolution = false
6: while B&B tree not empty or optimalSolution equals false do
7:     Sort B&B tree on the lower bound
8:     Choose the node with the minimal lower bound
9:     Include only generated paths that fulfill the branching
10:    optimalSolutionInNode = false
11:    if lower bound in node rounded up to the nearest integer equals T^Max then
12:        optimalSolution = true
13:        optimalSolutionInNode = true
14:    end if
15:    counter = 0
16:    while not optimalSolutionInNode do
17:        counter = counter +1
18:        Solve RMP and update dual values
19:        foundNewColumn = false
20:        for each vehicle k do
21:            Solve subproblem for k
22:            if subproblem for k returned some new columns then
23:                foundNewColumn = true
24:            end if
25:        end for
26:        if not foundNewColumn then
27:            optimalSolutionInNode = true
28:        end if
29:    end while
30:    if optimal solution in node is not feasible then
31:        Continue
32:    end if
33:    if optimal solution in node is integer then
34:        if optimal solution in node < T^Max then
35:            T^Max = optimal solution in node
```

```
        end if
        Continue
    end if
    if some arcs are serviced by more than one vehicle then
        Branch according to branching scheme
    else
        Branch on consecutive arcs for a vehicle with fractional solutions
    end if
    if counter > givenNumber then
        counter = 0
        Solve RMP as a MIP with all columns generated
        if MIP solution is better than T^Max then
            T^Max = MIP solution
        end if
    end if
end while
Return T^Max
```

5.4 Expanding the decomposition

Not included in the implementation of the column generation algorithm, is the constraints that prohibit the larger vehicles from performing U-turns. They were omitted as we wish to evaluate all of the available models on common ground: simply to produce routes in a network where operation synchronization is paramount. However, in order to disallow U-turns for the larger vehicles in the implemented column generation, one only need to change the subproblem. Specifically, one cannot extend a label in such a way that a U-turn is included in the route. Therefore, if label L_1's predecessor is in node i, L_1 cannot be extended back to node i. Additionally, a forth dominance criterion must be included as well, where one of two cases must be maintained. The first case is that if L_2 dominates L_1 (based on the three criteria in Section 5.2.5) and $\eta(\phi(L_1)) = \eta(\phi(L_2))$, then L_1 can be rejected. For the other case, if it exists two labels L_2 and L_3, which dominate L_1 (based on the three criteria in Section 5.2.5), and $\eta(\phi(L_2)) \neq \eta(\phi(L_3))$, then L_1 is dominated, as at least one of L_2 and L_3 can extend to all the nodes that L_1 can extend to.

Other turn restrictions can be implemented in a similar fashion.

Chapter 6

Hybrid Genetic Search with Diversity Control

In order to obtain solutions to instances of practical size, with sufficiently sized vehicle fleets, a heuristic approach has been considered. In this chapter we introduce a genetic algorithm based on recently developed methods within the VRP literature. An introduction to the overall algorithmic framework is given in Section 6.1. Representation of the solutions is considered in Section 6.2, and we elaborate on parent selection and crossover in Section 6.3. In section 6.4, the education phase is introduced, followed by an explanation of the survivor selection process in Section 6.5, which concludes the chapter.

6.1 Introduction

The proposed genetic algorithm (GA) is based on the paradigm that Holland (1975) is credited for introducing. The framework of genetic algorithms is built on an underlying idea of having a set of solutions (a population), and exploit mechanisms based on evolutionary processes to improve these solutions. The Hybrid Genetic Search with Adaptive Diversity Control (HGSADC) metaheuristic was introduced by Vidal et al. (2012) and has proved to be very efficient in many variations of the VRP. It includes a number of advanced features in terms of solution evaluation, generating and improving offspring, and population management, which contribute to its originality and high performance level. The algorithm developed and implemented in this thesis is mainly based on the

work by Vidal et al. (2012), and adapted to the MFARPOS. Due to specifics that are discussed in this chapter, we henceforth refer to our implementation as the Hybrid Genetic Search with Diversity Control - or simply HGSDC.

The general scheme of the metaheuristic is displayed in Algorithm 3. The algorithm iteratively evolves a group of individual solutions, namely the population. It successively applies a number of operators to select two parent individuals and combine them, yielding new individuals (offspring). The set of offspring are then enhanced by common local search procedures (education). Of particular interest is the evaluation mechanism proposed by Vidal et al. (2012), to select parents for mating (line 3 of Algorithm 3), and which individuals that are to survive to the next generation (line 9). The mechanism account for not only the solution cost, which is the norm within many evolutionary algorithm approaches, but also the contribution of diversity that the individual makes to the population. The effect of using a measure of diversity to decide whether an individual is fit to survive, and become a part of the population, is a paramount feature of the metaheuristic. It contributes to maintaining a wide range of solutions, thus decreasing the probability of convergence towards a suboptimal solution early on. With respect to diversity, a large part of the population is also replaced by random individuals every I^{Div} iterations without improvement, effectively diversifying the population and introducing new genetic material. The algorithm runs until a predefined number of iterations without improvement in objective value, I^n, have passed, or until a time limit is reached ($MAXTIME$ seconds).

Algorithm 3 HGSDC

```
1:  Initialize population
2:  while n iterations without improvement < I^n or running time < MAXTIME do
3:      Select parents
4:      Create offspring
5:      for each offspring do
6:          Educate offspring
7:      end for
8:      Add educated offspring to the population
9:      Select survivors
10:     if best solution not improved for I^Div iterations then
11:         Diversify population
12:     end if
13: end while
14: Return best solution
```

6.2 Solution representation

Solutions are represented as an array of identifiers. The array stores identifiers of arcs and information regarding which vehicle that service which arc in which order. Such a representation within genetic algorithms is often called a *chromosome representation*, or simply a *chromosome*. As all solutions to the MFARPOS involve two homogeneous fleets, each individual is represented by a set of two chromosomes: the *plowing truck chromosome*, which is an ordered list of tasks and trip delimiters for the plowing truck; and the *smaller vehicle chromosome*, which is an ordered list of tasks and trip delimiters for the smaller vehicles. An example chromosome is provided in Table 6.1, which represents a solution to the network found in Figure 6.1. The routes for the vehicles are found in Figures 6.2, 6.3, 6.4, and 6.5.

In the HGSADC developed by Vidal et al. (2012), the chromosomes are divided into routes by the use of the *Split* algorithm, as presented by Prins (2004). The Split algorithm takes as input a chromosome without trip delimiters, and returns the optimal distribution of tasks to the different vehicles. Only when the solution is dependent on a single fleet will the Split algorithm return the optimal solution. In the MFARPOS, where the objective function is to minimize the makespan, the best solution for the plowing trucks may not be the best solution for the smaller vehicles, and vice versa. The Split algorithm is therefore not optimal for this problem, and trip delimiters are a necessity for our chromosomes. A trip delimiter is a way to separate the vehicles' task list within the same fleet from one another. This is often accomplished by the use of a special symbol in the chromosome, to differ the trip delimiters from the tasks. These trip delimiters are distributed throughout the chromosome at random in the initial population, and are indirectly subject to movement by various insertions and swaps of tasks between vehicles in the Education phase (Section6.4).

Table 6.1: Example of a solution representation where the upper and lower rows show the chromosome for the larger and smaller vehicles, respectively. Each positive integer is a unique identifier of a lane or a sidewalk, while -1 is the trip delimiter. This example is therefore a solution to a problem with a network with a total of 20 arcs, two trucks and two smaller vehicles.

Plowing Truck	1	3	5	8	4	6	-1	2	7	9	10
Smaller Vehicle	11	13	15	17	18	-1	12	14	16	19	20

With exception of the trip delimiters, each value in the chromosome is referring to a service demanding arc in the graph. The chromosome comprises each vehicle's *task*

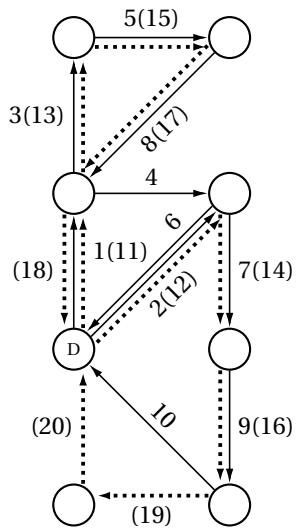

Figure 6.1: A road network for which the chromosome in Table 6.1 is a solution. The circles represent nodes (D = depot), solid and dotted lines represent the lanes and sidewalks, respectively. The numbers denote the arc identification numbers (sidewalks in parenthesis).

sequence. A task in this perspective is an arc, and the vehicle with a certain task in its task sequence is designated to service it. In graphs where each arc only has to be serviced once, the task sequence is unique in the way that no arc exists in more than one task sequence. If an arc needs to be serviced more than once, the algorithm copies the respective arc, and gives this copy a new unique identifier. The task sequence is not to be confused with the *routes*. The task sequence of a vehicle only indicates which arcs to service, and in what order. That is, deadheading is omitted from the task sequence. The route consists of all arcs that the vehicle is both servicing and deadheading, and therefore, it includes at least as many arcs as the task sequence. That is, the route includes the path from the depot to the first task in the task sequence, the tasks and the paths between each of the tasks, and the path from the last task in the sequence back to the depot. The paths and time used on the paths, between consecutive tasks in the sequence, are derived using the Floyd-Warshall algorithm (Floyd, 1962), which guarantees that the vehicles always deadhead the shortest path (with respect to time) between two given tasks. We note that for the heuristic, no artificial depots are needed, and so these are not included in the routes generated by the algorithm, nor in any illustration or discussion throughout this chapter.

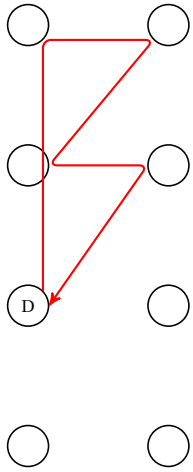

Figure 6.2: Illustration of the route for the first plowing truck on the network illustrated in Figure 6.1 according to the chromosome in Table 6.1. The vehicle services arc 1, 3, 5, 8, 4, and 6 back to the depot.

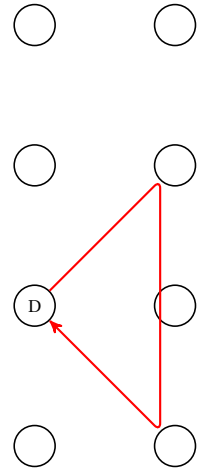

Figure 6.3: Illustration of the route for the second plowing truck on the network illustrated in Figure 6.1 according to the chromosome in Table 6.1. The vehicle services arc 2, 7, 9, and 10 back to the depot.

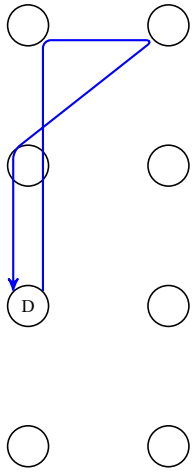

Figure 6.4: Illustration of the route for the first smaller vehicle on the network illustrated in Figure 6.1 according to the chromosome in Table 6.1. The vehicle services arc 11, 13, 15, 17, and 18 back to the depot.

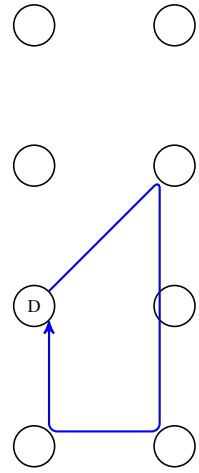

Figure 6.5: Illustration of the route for the second smaller vehicle in the network illustrated in Figure 6.1 according to the chromosome in Table 6.1. The vehicle services arc 12, 14, 16, 19, and 20 back to the depot.

When evaluating a solution, a deterministic simulation is performed on the vehicles' routes from and back to the depot. The algorithm starts with the plowing trucks, and stores the time each lane is served. Then, for each of the smaller vehicles, a check is performed to see if a synchronization constraint exists on the arc under consideration - and whether the corresponding lane is serviced prior to the time at which the smaller vehicle arrives at the task. If this is not the case, the smaller vehicle must wait for the truck to arrive. This waiting time is included when calculating the route time. When the simulation is over, the total time used by each vehicle is compared, and the makespan of the solution is stored along with the pair of chromosomes that it comprises.

6.2.1 Solution evaluation

The evaluation function seeks to associate each individual in the population with a value that is calculated with respect to the entire population. Such a function is generally referred to as a *fitness function*, and the function value is thus the *fitness* of the function argument. As noted by Vidal et al. (2012), such fitness functions are often based on the objective function of the problem at hand. The authors further note that the use of a fitness function solely based on the objective function is quite myopic with respect to the possible impact of the evaluation and selection properties on the *diversity* of the population. Therefore, they argue that the *fitness values* created by the fitness function should reflect two properties of each individual: 1) the fitness with respect to the objective function, and 2) how much the solution contributes to the diversity of the population.

We define the *Diversity Contribution*, $\triangle(P_i)$, of an individual P_i as the average distance to its n^{Close} closest neighbours, grouped in the set $\mathcal{N}_{P_i}$. A normalized Hamming distance $\delta^H(P_i, P_j)$ is used, based on the differences in the chromosome pair of the different solutions. Let $\{1,\dots,N\}$ be the tasks (including the trip delimiters) in the problem, and $\pi_{ku}(P_i)$ be the task for individual P_i at index u in the chromosome type k, where k is either a plowing truck or a smaller vehicle. The diversity contribution is computed according to

$$\triangle(P_i) = \frac{1}{n^{Close}} \sum_{P_j \in \mathcal{N}_{P_i}} \delta^H\left(P_i, P_j\right)$$

where

$$\delta^H\left(P_i, P_j\right) = \frac{1}{N} \sum_{k \in \mathcal{K}} \sum_{u=1}^{N} \mathbf{1}\left(\pi_{ku}(P_i) \neq \pi_{ku}(P_j)\right)$$

and $\mathbf{1}$ is a binary function that returns 1 if the statement is true, and 0 otherwise.

Let $\mathcal{P}$ be the set of individuals in the population, and *fit(P_i)* and *dc(P_i)* represent the rank (from 1 to $|\mathcal{P}|$) of an individual P_i in population $\mathcal{P}$ with respect to its makespan and diversity contribution, respectively. The *biased fitness* function, *bf(P_i)*, combines the makespan and the diversity contribution, and is given by

$$bf(P_i) = fit(P_i) + \left(1 - \frac{n^{Elite}}{|\mathcal{P}|}\right) dc(P_i)$$

where n^{Elite} is a parameter indicating the number of *elite* solutions in the population. An elite solution is one of the n^{Elite} best solutions with respect only to the function value of *fit(P_i)*. When the biased fitness function is defined as above, all of the elite solutions will survive to become a part of the next generation. Vidal et al. (2012) provide a proof for this, which is omitted here. We emphasize that the parameter $\frac{n^{Elite}}{|\mathcal{P}|}$ can be viewed as a factor which controls the trade-off between regular fitness (calculated from the makespan) and the diversity fitness.

Vidal et al. (2012) allow infeasible solutions, and introduce adaptivity as a way to increase the penalty of infeasibility over time. While this is not implemented in our heuristic, as infeasible solutions simply are not created (see Section 6.2.2 for details), we introduce an alternative approach to adaptivity as an option. Given the aforementioned definition of the n^{Elite} parameter, a low parameter value increases the value (or importance) of diversity, while high n^{Elite} values regular fitness, and thus elitism. Let r be the number of iterations since the last improvement in makespan. When this value is low compared to the maximum number of iterations, I^n, the search should favor diversification, allowing a more extensive search in the search space, and escape a potential local optimum. When r is high compared to I^n, the search should favor intensification of the search, exploring the more promising regions of the population to find a better solution. If a better solution is found, the r counter is reset, and also the n^{Elite} is set to its initial value. The parameter is therefore changed dynamically according to

$$n^{Elite} = \left\lfloor \frac{r \times |\mathcal{P}|}{I^n} \right\rfloor$$

which allows the search to adapt to how long it has been since it last found an improving solution. This notion of adaptability is tested in Chapter 8 against a static parameter value. Note here that when there is a need to differentiate between the models, we refer to them as the *Basic HGSDC* and the *HGSDC with adaptiveness*.

6.2.2 Search space

Both Vidal et al. (2012) and Borthen and Loennechen (2016) argue that optimal solutions lies on the boundary of feasibility, and that allowing infeasible solutions enhances the performance of the search. Infeasible solutions with respect to vehicle capacity constraints and maximum route travel time constraints are therefore allowed. However, breaking other important constraints in their problems is disallowed entirely. The problem studied in this thesis has some fundamental differences with these vehicle routing problems. First of all, the MFARPOS does not operate with vehicle load, and therefore no vehicle load constraints exists. Secondly, this problem seeks to minimize the makespan, which means that the model seeks to minimize the route with the longest travel time in each solution. Solutions breaking this constraint will therefore not be candidates for good solutions in the case of our problem.

There is, however, an option to allow solutions that break one or more of the synchronization constraints, which is one of the core features of the MFARPOS. Those constraints are handled according to the elaboration in Section 6.2, emphasizing that the smaller vehicles simply need to wait if a lane with an associated synchronization constraint has not been serviced when the smaller vehicle arrives at the sidewalk. Let us define the time each smaller vehicle k has to wait to service arc (i, j) as w_{kij}. The total waiting time W_k for vehicle k can be expressed by

$$W_k = \sum_{(i,j)\in\hat{A}^S} w_{kij}$$

This means that the vehicle can just wait W_k time units in the depot before it starts its route, and then complete it without any further waiting. If route r of vehicle k has a traversal time of τ_{rk}, then the total time used by vehicle k becomes

$$T_k = \tau_{rk} + W_k$$

The total waiting time therefore becomes a "cost" of infeasibility. However, this cost is more than merely a symbolical cost assigned because the solution is infeasible. This is actually the minimum waiting time one can add to make the solution feasible. Speaking of infeasible solutions with respect to the synchronization constraint therefore makes no sense, as every solution can be made feasible by imposing waiting to the smaller vehicles. In fact, there is even a possibility that all *optimal* solutions include some waiting time. As infeasible solutions with respect to the synchronization constraints are

implicitly considered, and that the infeasibilities considered by others to improve their heuristic do not apply to our problem, we have chosen to simply avoid evaluating infeasible solutions altogether.

6.2.3 Turn restrictions

The implementation of the heuristic comes with the option to disallow U-turns. While not a primary target for our study, this is implemented as a static penalty for each U-turn the vehicles perform in the solution. Let u_{kij} be 1 if vehicle k traverses an arc (i, j) and then directly continues on arc (j, i), and 0 otherwise. Let P^U be the U-turn penalty cost. The total penalty cost for vehicle k becomes

$$P_k = \sum_{\substack{(i,j)\in\mathcal{A}_k \\ |(j,i)\in\mathcal{A}_k}} P^U u_{kij}$$

When included in the model, the total penalty cost is added to the total time of the vehicle's route, which becomes

$$T_k = \tau_{rk} + W_k + P_k$$

In addition to the U-turn restriction, it is also possible to implement other turn restrictions. The same procedure as above could be utilized, adding a high penalty for each illegal turn. One example of this is the *left turn* restriction, which prohibits the vehicles to take left turns. In this example, one would need to make binary relations with each arc and its left turn, if it exists. As these relations are not built into the graph structure of the problem, this would have to be implemented as well.

As explained, the U-turn penalty is optional, and for the rest of the chapter, we assume that the penalty cost does not apply.

6.3 Parent selection and crossover

The offspring generation scheme of the HGSDC selects two individuals, $P1$ and $P2$, and creates two offspring, $O1$ and $O2$. Parent selection is performed through a *binary tournament*. Binary tournament is a procedure in which two individuals are randomly chosen (here with uniform distribution) from the entire population. They are ranked according to their biased fitness, where the winner (one of the individuals) is the one

with the best biased fitness. This is done twice in order to obtain both parents. We note that in this very process, the individual with the lowest biased fitness in the population has a probability of 0 to be chosen as a parent. This can be corrected by trivial methods, but are seen as insignificant in the grand scheme of things, and are therefore not corrected in any way here. We emphasize that indirectly leaving the individual with the worst biased fitness out of the parent selection process is equal to, in this step, viewing the population as simply having one less individual.

To create a feasible individual without the need of repairing it, the mating process used by the HGSDC for the MFARPOS is the Partially Mapped Crossover (PMX). PMX was introduced by Goldberg and Lingle (1985), and is widely used for combinatorial optimization problems. Ting, Su, and Lee (2010, p.1880) state that "...the PMX is one of the most popular and effective crossovers for order-based genetic algorithms, especially the TSP". In our algorithm, the crossover is given two *input* chromosomes of the same vehicle type, and it returns an *output* chromosome. An *individual/offspring*, on the other hand, is the combination of a smaller vehicle chromosome and a plowing truck chromosome.

The Partially Mapped Crossover is described in Algorithm 5. The algorithm gets two chromosomes of the same vehicle type as input, called *Input1* and *Input2*. Initially, all the trip delimiters are removed from these chromosomes, such that the each input consist only of one task list. Then, the PMX proceeds by taking a random substring from each of the inputs. The index range where these substrings are located is equal for both inputs, and is referred to as the *swath*. The substring from Input1 is copied to the output chromosome, before a second process iteratively searches through Input2's substring, in order to determine the element values (the arc identification numbers) that are currently not in the output. Several operations are performed to assign the missing swath values to the output. In order to make sure that all service demand is accommodated, the last step is to copy all of the remaining element values (equals the values not included in any of the substrings) from Input2 to the output, and insert the trip delimiters from Input1. Note that this process only creates one chromosome, and as each individual has both a smaller vehicle chromosome and a plowing truck chromosome, this process must be run twice to create an individual. This is summarized in the *Mating* algorithm (Algorithm 4), which runs the Partially Mapped Crossover once for each vehicle type. To create two individual offspring, the two parents ($P1$ and $P2$) are swapped in the Mating algorithm. This means that the Mating algorithm is run twice for each chosen pair of parents, with a newly generated swath for every child chromosome created,

and returning two offspring. The following example is made to visualize the process of the PMX algorithm. Note that the example only creates one output chromosome.

Algorithm 4 Mating(P1, P2)

```
Input1 = Plowing truck chromosome from P1
Input2 = Plowing truck chromosome from P2
PTOutput = Partially Mapped Crossover(Input1,Input2)
Input1 = Smaller vehicle chromosome from P1
Input2 = Smaller vehicle chromosome from P2
SVChild = Partially Mapped Crossover(Input1,Input2)
Combine PTOutput and SVChild to create O1
return O1
```

Algorithm 5 Partially Mapped Crossover(Input1, Input2)

```
Remove all trip delimiters
Get a random range from Input1
Copy this range to Output1
for each value in Input2 in the same range as the swath in Input1 do
    if value is not in Output1 then
        next = value
        tempNext = value
        while next is not in Output1 do
            Find the value x at the same index as tempNext in Input1
            Find the index i where x resides in Input2
            if index i is not in the swath then
                Copy next to index i in Output1
            else
                tempNext = value at index i in Input2
            end if
        end while
    end if
end for
Copy everything else from Input2 to Output1
Insert the trip delimiters at the same indices as in Input1
```

Table 6.2 presents the first step of the Partially Mapped Crossover, on a very simplified example. The trip delimiters (marked in red) from both input chromosomes are removed. These are found at index 5 and index 6 in Input1 and Input2, respectively. These indeces are stored for insertion of the trip delimiters in the last step of the PMX.

Table 6.2: Partially Mapped Crossover example, stage 1.

Input1	1	3	5	6	-1	4	2	7	8
Input2	5	6	2	1	3	-1	4	8	7

Table 6.3 presents the next step of the Partially Mapped Crossover. A random index sequence - the swath - is selected from the inputs. In the example at hand, the swath is a list of the consecutive values from 2 to 4. The swath is marked with thick borders in the table. All of the element values at the index sequence of the swath in Input1 (marked in red) is copied to the same index numbers in Output1.

Table 6.3: Partially Mapped Crossover example, stage 2.

Input1	1	3	5	6	4	2	7	8
Input2	5	6	2	1	3	4	8	7
Output1	_	3	5	6	_	_	_	_

Going through the swath values of Input2, the first element that does not already exist in Output1 is located. In the example, this element has a value of 2, and is on index 3 in Input2. Algorithm 5 proceeds to step 9, and finds the value 5 at index 3 in Input1. Step 10 follows, where the index of the value 5 in Input2, which is 1, is located. This index is not in the swath, and therefore the value of $next(=2)$ is copied to index 1 in Output1. This step is summarized in Table 6.4, and the values in question are marked in red.

Table 6.4: Partially Mapped Crossover example, stage 3.

Input1	1	3	5	6	4	2	7	8
Input2	5	6	2	1	3	4	8	7
Output1	2	3	5	6	_	_	_	_

The next (and final) value from the swath range of Input2 not existing in Output1, is 1. This is found on index 4 in Input2. Step 9 and 10 are now initiated: the value at index 4 in Input1 is 6, which is placed at index 2 in Input2. As this index is a part of the swath, and therefore occupied in Output1, Algorithm 5 proceeds to step 14, and repeats

the while-loop with 6 as a placeholder, $tempNext$, for 1. This procedure is visualized in Table 6.5, and the values in question are marked in red.

Table 6.5: Partially Mapped Crossover example, stage 4.

Input1	1	3	5	6	4	2	7	8
Input2	5	6	2	1	3	4	8	7
Output1	2	3	5	6	_	_	_	_

The placeholder, 6, is located at index 2 in Input2. The value at index 2 in Input1 is 3, which is placed at index 5 in Input2. As this index is not a part of the swath, and not yet occupied by an element in Output1, step 12 follows, where Algorithm 5 copies the value of $next(= 1)$ to index 5 in Output1. The result is shown in Table 6.6, and the values in question are marked in red.

Table 6.6: Partially Mapped Crossover example, stage 5.

Input1	1	3	5	6	4	2	7	8
Input2	5	6	2	1	3	4	8	7
Output1	2	3	5	6	1	_	_	_

As there no longer exists any elements in the swath of Input2 that are not already in Output1, Algorithm 5 proceeds to step 19, where all of the remaining elements is copied from Input2 to Output1 at the same indexes as in Input2. The resulting list is shown in 6.7 and the values in question are marked in red.

Table 6.7: Partially Mapped Crossover example, stage 6.

Input1	1	3	5	6	4	2	7	8
Input2	5	6	2	1	3	4	8	7
Output1	2	3	5	6	1	4	8	7

The last step of Algorithm 5 inserts the trip delimiter (marked in red) at the same index in Output1 as they were located in Input1. In the example, this was at index 5. The elements at higher index numbers than the trip delimiters are shifted towards the right, in order to conserve all tasks in the chromosome. Table 6.8 shows the resulting chromosome after inserting the trip delimiter, and this completes the Partially Mapped Crossover.

Table 6.8: Partially Mapped Crossover example, stage 7.

Output1	2	3	5	6	-1	1	4	8	7

6.4 Education

An education operator is applied with a given probability of η^{Edu}, to improve the quality of the offspring's solution. This education procedure goes beyond the classical genetic algorithm concepts of random mutation and enhancement through hill-climbing techniques, as it includes a number of local search procedures based on neighborhoods specific for this MFARPOS.

Two sets of local search procedures are defined. Eight route improvement procedures, which are dedicated to optimize the solutions based on the neighborhood, constitute the first set. The second set is based on four new route improvement procedures, all of which are stochastic. Let $k(u)$ represent the vehicle containing arc u in its given task list, and (u_1, u_2) identify the partial route from u_1 to u_2. Define the neighborhood of arc u as all arcs which start in the end node of arc u. Let v be a neighbor of u, and x and y the successors of u in $k(u)$ and v in $k(v)$, respectively. The first route improvement phase iterates, in random order, over each arc u and each of its neighbors v, and evaluates the implication of the following moves:

(M1) Remove u and place it directly after v

(M2) Remove u and x, then place u directly after v and x directly after u

(M2) Remove u and x, then place x directly after v and u directly after x

(M4) Swap u and v

(M5) Swap u and v, then place x directly after u

(M6) Swap u and v, then swap x and y

(M7) Swap x and v

(M8) If $k(u) \neq k(v)$, replace (u, x) and (v, y) by (u, y) and (x, v)

The first three moves correspond to insertions, whereas M4 through M7 generally are known as *swaps* in the literature. M1 through M7 can be applied independently on the same or different routes, while M8 is an *interroute* swap. The effects of the moves are examined in random order, the first yielding an improvement being the one that

is implemented. Improvement here being either 1) a reduced makespan, or 2) that the sum of the involved vehicles' total time is less than the sum of their total time before the move (with equal makespan). The first route improvement phase stops when all possible moves have been successively tried without any of them yielding an improvement.

While the first route improvement phase is based on neighborhoods of the arcs, and the assumption that an optimal solution will be subject to very little deadheading, the other phase is an acknowledgement to the fact that the optimal solution *may* involve some deadheading, and enables the search to find these solutions as well. The second route improvement phase only starts if the first phase shows to be unsuccessful. This phase iterates over each arc i, randomly select arcs j, l and m, and find the task list in which they are included ($k(i)$, $k(j)$, $k(l)$ and $k(m)$, respectively). It then evaluates the effect of the following moves:

(M9) Remove arc i and add it after arc j

(M10) Remove arc i and arc l, and add them after arc j and arc m, respectively

(M11) Swap arc i with arc j

(M12) Swap arc i and arc l, with arc j and arc m, respectively

The incentive to iterate over the entire set of arcs, is to ensure that all arcs have been tried in at least one move, increasing the chances of finding a good move. M10 and M11 are insertion moves, the second being a double insertion. M12 and M13 are swaps, with the final move being a double swap. Also here are the effect of the moves examined in random order: the first yielding an improvement being the one that is implemented. Improvement here meaning the same as above. The second route improvement phase stops when all possible moves have been tried without success.

6.5 Population management

Population management consists of several phases, whose purpose is to identify and propagate good solutions, enhance population diversity, and provide the means to do an efficient and thorough search. Four main phases can be derived for the management of the population: Initialization, Offspring introduction to the population, Survivor Selection, and Diversification. These are discussed in the following. Note that the initialization phase occurs only once, at the very beginning of the heuristic, in contrary to the others, which happen routinely.

6.5.1 Initialization

To initialize the population $\mathcal{P}$, μ individuals are created randomly by generating random task lists and routes. They are then added to the set of solutions which constitutes the population. All of the individuals in $\mathcal{P}$ are subject to the survival selection phase later on.

6.5.2 Offspring introduction to the population

After going through the education procedure, all λ offspring are entering the phase where they are introduced to the population. This phase assesses each of the offspring's diversity contribution to the population. This is closely related to the solution evaluation part of the heuristic, as discussed in Section 6.2.1. In addition to evaluating diversity, the algorithm also explicitly removes all *clones* from the population. A clone is a *new* individual that is identical to an existing individual in the population. The reason for removing clones is that they do not contribute to the diversity of the population, and increases the chances of premature convergence of the search. After assessing the individuals with respect to diversity and removing the clones, the individuals are added to the population. We note that two individuals may contain the exact same task lists in their chromosomes, but in different order. These are not considered as clones, as their chromosomes differ and may be good candidates for the mating process.

6.5.3 Survivor Selection

Premature convergence of the population is a major concern and challenge with population based algorithms. The HGSDC therefore includes a mechanism to deal with this problem, by giving individuals which contribute more to the diversity of the population a higher probability of survival to the next generation.

After all offspring have been introduced to the population, there are a total of $\mu + \lambda$ individuals in $\mathcal{P}$, minus the possible clones that have already been removed. As long as the population has a size greater than μ, the algorithm finds the individual with the worst biased fitness, and removes it from the population. However, the algorithm will never remove the n^{Elite} best solutions, with respect to the objective value, from the oversized population. This is to ensure enough intensification to the search, as these solutions may result in more promising solutions with more education, and good parent candidates for the mating process. A pseudocode for the Survival Selection phase is found in Algorithm 6.

Algorithm 6 Survivor Selection

```
1: while population > μ do
2:     Evaluate and sort the population from best to worst biased fitness
3:     Remove the member of the population with the worst biased fitness
4: end while
```

6.5.4 Diversification

A final important aspect of population management is the diversification of the population. When I^{Div} successive generations without improvement have occurred, the algorithm "reboots" and removes all but the $\lfloor \frac{\mu}{3} \rfloor$ best individuals, with respect to biased fitness, from the population. It proceeds by creating random chromosomes in a similar fashion as when the heuristic is initialized. The goal of this diversification is to introduce new genetic material to the population, and prevent it from getting stuck in a local optimum. These newly added individuals are not likely to improve the makespan objective once they are added, depending on parameter values etc. However, they may contribute positively by being good mating options, thus reviving the search in a more diverse fashion. The denominator in the fraction above (3) follows the example sat by Vidal et al. (2012).

Chapter 7

Generation and Characteristics of Test Instances

All data instances used in this thesis are based on fictitious road networks, with characteristics that are designed to mimic those one generally find in urban and residential areas. As many cities, Trondheim included, share common features, such as a square-grid plan layout and intersections generally consisting of four roads going in and out, this is incorporated into the instances, which are generated by a program with random components determining locations and traversal times of lanes and sidewalks.

In Section 7.1 we introduce the instance generator that has been developed and implemented to generate data sets. Discussion is centered around functions of the program, how the instances are generated, with a brief view on parameter settings, and the characteristics that the instances are designed to inherit. The different sets of instances that have been generated, and their application within this thesis, are described in Section 7.2.

7.1 Instance generation

The code is written in MATLAB, and has a running time of only a couple of seconds for data instances of sizes beyond the scope of this thesis. It is designed to stochastically generate instances that are based on block structures and intersections, with most blocks being squares and triangles (as found in cities). In the instances generated, no two road segments form a cross without the existence of a vertex in the intersection.

This is equal to saying that bridges and tunnels crossing roads are omitted in these instances, and that the corresponding graph is planar. Additionally, park-structures and other pedestrian paths are added, such that there can exist a *sidewalk* without there being an adjacent lane. However, for simplicity, all nodes in the final graph are endpoints of at least two lanes, such that no connections of only sidewalks are presented. We emphasize that although the underlying design of the instances is based on roads and blocks, a node in the graph may not be an intersection, and although the graphical representations of lanes and sidewalks are straight, they may in reality be curved and longer or shorter relative to the other arcs in the network. Thus, in this chapter nodes represent connections as they are defined in the Chapter 3, and the triangle inequality does not count for the instances generated (even if the associated traversal times would represent distance). Also, due to the focus of the computational study, instances generated have a maximum of one lane in each direction in a road segment (between two nodes). The instance generator can easily be modified to accommodate this, whereas all models developed already support this feature.

The steps of the instance generator are summarized below, and are discussed in detail. Note that what is referred to as a pseudo-Markov process, do not fulfill all of the requirements for a Markov process in the way we model states and transitions. That is, the process is not completely independent with respect to time. This is elaborated on in the description of step 4.

1. Initialize with parameters
2. Generate Markov chain
3. Generate initial road segments, and their associated service time
4. Run pseudo-Markov simulation
5. Clean traversal time matrix
6. Log and name active nodes
7. Add sidewalks to the network, and generate service times
8. Generate deadheading times for both vehicle types
9. Add artificial depots
10. Write TikZ-code to file for graphical representation

11. Format data and write files for solvers

Step 1 is where the user sets the parameters $size$ and $nIter$. The $size$ parameter is an odd integer, indicating the dimensions of the underlying network, which is set to have $size \times size$ nodes. When running the Markov simulation, however, not all of the nodes are necessarily being used, thus this refers to an upper bound for the number of nodes in the resulting graph. The two artificial depots are added to the network in step 9. The $nIter$ parameter corresponds to the number of iterations in the pseudo-Markov simulation, which generally should be higher for a higher value of $size$.

Step 2 is where the Markov transition matrix is generated. A $size^2 \times size^2$ matrix is constructed, where each element at index (i, j) corresponds to the probability that j is the next state of the process when the current state is i. Notice here that allowed values for i and j correspond to two nodes that are within reach of one another in the underlying network - that is vertically, horizontally or diagonally. All other entries in the matrix are set to zero. To avoid two diagonal paths to overlap, a random process is initiated to determine which of the two transition matrix elements that is given the non-zero value. In addition the transition matrix always allows for traversal in two directions. If for example the entry at $(i, j) > 0$, then so is the entry at (j, i).

In step 3, a $size^2 \times size^2$ matrix is initialized (with zeroes in all entries), which is used to store service times between nodes. Additionally, a set of initial arcs is constructed. These arcs form a large cross, mimicking two main roads going through the road network. These are two-way road segments which can optionally be set to *definitely* have sidewalks on both sides. These arcs form a basis for the instances generated, but are in step 5 subject to removal if they make the network unable to be serviced without the necessity of U-turns for the large vehicles. The initial arcs are generated at the same location for all instances with the same $size$-parameter. On the other hand, the service- and deadheading times for these arcs are always determined stochastically, and therefore usually vary between instances. Service times are always generated when an arc is added to the graph (in fact, the non-zero service time is what indicates there being an arc), and therefore the service times of these initial arcs are generated in this step. The road network generated in this step is illustrated in Figure 7.1. Solid and dotted lines represent lanes and sidewalks, respectively.

Step 4 is where all of the remaining lanes are added to the road network. A simulation is run, where a single vehicle begins in the depot (located at the endpoint of the initial cross), and moves to one of the neighboring connections (nodes) with the probability given by the transition matrix. The probability of moving to one node from

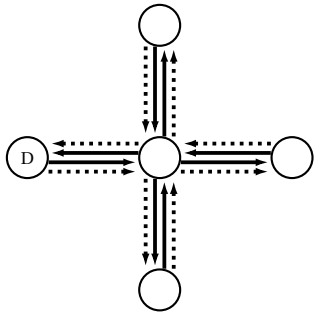

Figure 7.1: The generated road network after the first set of arcs and sidewalks are added.

another is constant, with one exception in each iteration - which is what distinguishes this process from a true Markov process (which is time independent). The explanation for this change in the transition probability stems from wanting the plowing trucks to be able to service the entire road network without having to perform U-turns. Indeed there exist very few dead ends in cities where there is demand for snow plowing vehicles immediately after a snowfall, and so they can be omitted from the underlying road network generated. Therefore, in each iteration of the simulation, the vehicle is denied access to the node whence it came. Please note that this could have been modeled as a true Markov process by adding many more states, with more information included in each state. However, this would cause the nodes to only be implicitly associated with states in the Markov process. For the sake of the implemented algorithm, there is a one-to-one correlation between nodes in the road network and states in the pseudo-Markov process. The network generated to this point is illustrated in Figure 7.2.

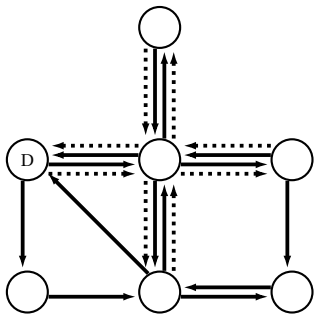

Figure 7.2: The generated road network after step 4, where the remaining lanes are added.

In each of the $nIter$ steps in the pseudo-Markov process, the next node (state) of the

simulated vehicle is chosen stochastically. A check is performed to determine whether the path from the current node to the chosen node already exist. If so, the vehicle changes position (that is, a variable indicating the current node is set equal to the variable indicating the next node). If the path do not exist, an arc is also added to the road network by stochastically determine the service time between the nodes, and storing it in the service time matrix. When $nIter$ iterations are completed, the vehicle is *forced* to return to one of the nodes in the cross that was initially constructed.

Due to the generation of the initial nodes and arcs, and the last step of the simulation (where the vehicle simply stops in a random node), some dead ends might exist in the network. That is, there might exist nodes that only can be accessed from a single other node *and* only be left by returning to the node from where the truck came. All dead ends are removed in step 5 by setting the traversal times to and from these nodes to zero (which indicates that they are inactive in the resulting graph). By removing dead ends, new ones might be generated. Therefore, this step is solved by using a recursive function, where the deepest step simply confirms that no more dead ends exist, and returns the input matrix. Figure 7.3 illustrates the network after the dead end in Figure 7.2 has been removed.

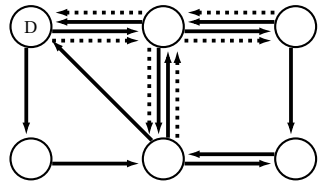

Figure 7.3: Road network after the dead end has been removed.

In step 6, all of the active nodes, which are the initial nodes and the nodes visited in the pseudo-Markov simulation minus the dead ends, are logged in a separate matrix. Each node is given a unique name (an integer) for identification. All nodes that cannot be visited from the initial nodes are from this point on neglected, and will never be added to the network.

Sidewalks are added to the network in step 7. A service time matrix for sidewalks is initiated. Then, a function iterates over all lanes, stochastically (with an adjustable probability parameter) determining which lanes should be accompanied by a sidewalk, before adding a randomly generated service time to the matrix. A synchronization criteria will exist for all of these sidewalks. Additionally, sidewalks are randomly added where no lanes exist between nodes that already exist in the network. The service time

of a sidewalk is generally set to be roughly 20% longer than that of the associated lane (if such a lane exist), with an extra random term. Figure 7.4 shows the example road network after the remaining sidewalks have been generated.

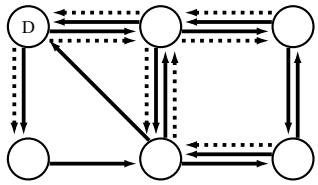

Figure 7.4: Road network after step 7, where the remaining sidewalks are added.

In step 8, two new matrices are generated, indicating the deadheading times for the larger trucks, and the smaller vehicles, respectively. The larger trucks are able to traverse only the lanes, whereas the smaller vehicles can deadhead both lanes and sidewalks. As the main restriction to the speed at which lanes can be driven is the speed limit, all of the vehicles have the same deadheading time between connections where there exist a lane, in addition to it always being shorter than the associated service time.

Artificial depots (for both starting and ending) are added to the road network in step 9. Graphically, the artificial depots are located to the left of the road network, only connected to the leftmost node in the initial main roads, which always serve as the true depot. Thus, instances with equal $size$ parameter have the artificial depots at the exact same location with respect to the graphical representation. The complete road network in our example is illustrated in Figure 7.5, where the nodes marked with D_s and D_e are the artificial start and end depot, respectively. This network corresponds to that in instance 8N21A from Test set 1, which are discussed in Section 7.2.2.

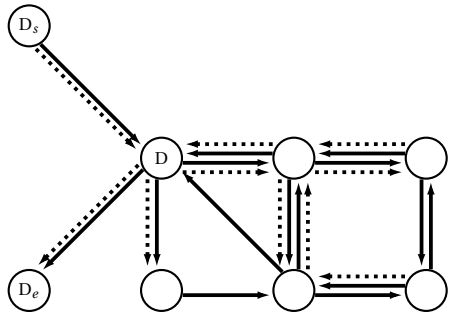

Figure 7.5: The complete generated network, with the artificial depots included.

In step 10, the code for a graphical representation of the road network is constructed. A function takes in all the data generated thus far and outputs a .txt-file with TikZ-

commands that can be used with the TikZ package - a powerful tool for including graphic elements in LaTeX, the typesetting system in which this thesis is written. If certain features are unwanted in the road network, a graphical inspection is in this case paramount to carefully oversee the matrices generated.

Final formatting of the matrices is performed in step 11. This is where the data are written to files that serve as the input for the implementation of both the exact solution methods, and the genetic algorithm. Two final parameters, namely the number of trucks and smaller vehicles, are included *by hand*. This is so, as we want to solve the same data instance with different fleet sizes.

7.2 Test instances

The test instances used in this thesis are now described. Several sets of instances have been generated, in order to serve different purposes. The characteristics of each instance are described by the number of nodes in the underlying graph (including the artificial depots), the number of lanes, sidewalks, and synchronization relations (the *Sync*-column in the tables listing the instances), all of which are effectively determined stochastically by the instance generator. Additionally, the sizes of both vehicle fleets, $|\mathcal{K}^L|$ and $|\mathcal{K}^S|$, which are chosen after the network is generated, are also included. The parameters *Legs* and *Max time* ($12 \times Legs$) refer to the number of legs that each route is allowed to have and a predefined upper bound for the makespan, respectively, in the arc-flow model. The Branch-and-Price algorithm uses the max time parameter, while the genetic algorithm does not make use of these parameters. The values for *Legs* and *Max time* are simply chosen manually by a heuristic not discussed further, due to the scope of the thesis. It shall be mentioned though, that this heuristic has no guarantee of making the number of legs optimal, nor feasible for the optimal solution with respect to makespan. Initial testing, however, shows that $Legs$ and *Max time* are sufficiently large for Test set 1, and cause no problem for the purpose of this thesis.

7.2.1 Test instances for parameter calibration

The genetic algorithm presented in Chapter 6 has several parameters that needs to be calibrated in order to obtain the best outcome. In this thesis, we primarily study the makespan objective, and thus the parameters should be calibrated thereafter. It should come as no surprise that the heuristic's parameters would be somewhat different for other objectives, such as shortest total time (often referred to as the cost objective).

Details regarding the parameters, how the calibration is performed, and the outcome of it, is given in the computational study in Chapter 8. However, to ensure an unbiased setting of the parameters, a separate set of test instances was generated for this very purpose. The main characteristics of the set, which is named Test set 1, are summarized in Table 7.1.

Table 7.1: Characteristics of the test instances in Test set 1.

Test case	**Nodes**	**Lanes**	**Sidewalks**	**Sync**	$\lvert\mathcal{K}^L\rvert$	$\lvert\mathcal{K}^S\rvert$
27N135A	27	66	69	54	3	3
27N138A	27	71	67	57	4	4
51N304A	51	160	144	120	5	5

The set consists of three instances, all within the range of medium to large sized. The smallest instance, with respect to the number of arcs, is also associated with the smallest number of vehicles in the fleets. The second instance, which merely has three more arcs than the smallest one, has an extra vehicle in both fleets. The largest instance, which has more than twice the amount of arcs than its predecessors, also has the largest fleets, with five vehicles to serve the lanes, and five vehicles to serve the sidewalks. Note that since the number of vehicles in both fleets are equal for a given instance, and that the number of lanes are roughly equal to the number of sidewalks; it is likely that it is the fleet of smaller vehicles that is primarily the constraining factor for the makespan. This is both due to the fact that sidewalks on average take longer time to service, and that the sidewalks are more scattered (with no guarantee to be consecutively connected) in the road network. This is a common feature for most instances in all of the sets generated to test our models. It could be argued that Test set 1 should involve other features, such as networks with very few sidewalks, vehicle fleets of different size, or other tendencies found in urban or residential areas. However, due to the number of parameters that need be set, and in order to do so consistently, we found it reasonable to focus on medium to large instances with a large degree of synchronization relations, and varied fleets of significant size.

7.2.2 Test instances to compare the models

Following the parameter calibration in Chapter 8, we compare all of our available models. That is, not only the B&P and the genetic algorithm developed for this thesis are tested, but also the mixed integer program (which is the implementation of the arc-

flow formulation). In order to do this we relax the U-turn constraints for the plowing trucks, such that there is no turn restrictions for any of the vehicles. This is so, since little effort has been given to formulate turn restrictions in the B&P algorithm. However, we do argue that such a comparison between the models provides a good indication of their performance differences. When relaxing these constraints, we refer to the models as *basic models*. The test instances for this comparison study are summarized in Table 7.2.

Table 7.2: Characteristics of the test instances in Test set 2.

Test case	**Nodes**	**Lanes**	**Sidewalks**	**Sync**	$\|\mathcal{K}^L\|$	$\|\mathcal{K}^S\|$	**Legs**	**Max time**
8N21A	8	13	8	8	2	1	16	192
8N28A	8	14	14	12	2	2	16	192
9N27A	9	14	13	9	2	2	16	192
10N38A	10	19	19	15	3	3	16	192
11N33A	11	20	13	12	3	2	16	192
12N37A	12	23	14	14	3	2	20	240
13N46A	13	27	19	16	3	3	20	240
14N46A	14	28	18	16	3	3	20	240
15N44A	15	25	19	14	3	3	20	240
16N52A	16	27	25	17	3	3	20	240

Due to the complexity of the problem, these instances are of small to medium size, incrementally increasing node by node (after which they are sorted). In order to study the behavior of the different models for different scenarios of importance to the problem under study, the ratio of sidewalks to lanes varies, whereas the ratio of lanes to nodes is about the same for all of the instances. The fleet sizes are also increasing as the network increases, the largest instances having three vehicles in both fleets. It is expected that both of the exact implementations are able to solve the smallest instances to optimality, while having a hard time finding good solutions for the largest instances.

7.2.3 Test instances for the further evaluation of the exact models

In order to further evaluate the performance of the best B&P model on medium sized instances, Test set 3 was generated. The characteristics of the instances are listed in Table 7.3. The number of nodes increases throughout the set. Three of the instances contain more sidewalks than lanes, and so the number of smaller vehicles is greater than the corresponding size of the fleet of plowing trucks. This is in order to increase

the probability that the optimal routes for the two vehicles types are about equal.

Table 7.3: Characteristics of the test instances in Test set 3.

Test case	Nodes	Lanes	Sidewalks	Sync	$\lvert\mathcal{K}^L\rvert$	$\lvert\mathcal{K}^S\rvert$	Legs	Max time
22N99A	22	56	43	36	3	3	26	312
22N108A	22	50	58	42	3	4	26	312
24N123A	24	61	62	50	4	5	30	360
26N116A	26	66	50	42	4	4	30	360
27N139A	27	68	71	55	4	5	30	360

As studying the exact models on Test set 2, it uncovered some interesting results for dense graphs with many synchronization relations, Test set 4 was developed to further examine how the exact models handle these. The characteristics of this set are presented in Table 7.4. All instances in Test set 4 have 11 nodes, where the number of arcs increases throughout the set. To keep the instances as identical as possible, the number of vehicles is equal with respect to the fleet type, in the three first and the three last instances.

Table 7.4: Characteristics of the test instances in Test set 4.

Test case	Nodes	Lanes	Sidewalks	Sync	$\lvert\mathcal{K}^L\rvert$	$\lvert\mathcal{K}^S\rvert$	Legs	Max time
11N52A	11	24	28	23	3	4	16	192
11N54A	11	25	29	25	3	4	16	192
11N55A	11	26	29	24	3	4	16	192
11N56A	11	25	31	25	4	5	16	192
11N59A	11	27	32	27	4	5	16	192
11N60A	11	28	32	28	4	5	16	192

7.2.4 Test instances for the genetic algorithm

To further evaluate the performance of the genetic algorithm on larger instances, Test set 5 has been developed. The set includes nine instances, grouped into three subsets where the graphs have the same number of nodes, and equal size of the vehicle fleets. Although the parameters of the instance generator remained constant with respect to each of these subsets, the number of arcs varies slightly throughout the instances. The characteristics of Test set 5 are presented in Table 7.5.

Table 7.5: Characteristics of the test instances in Test set 5.

Test case	**Nodes**	**Lanes**	**Sidewalks**	**Sync**	$\lvert\mathcal{K}^L\rvert$	$\lvert\mathcal{K}^S\rvert$
51N373A	51	174	199	168	6	8
51N389A	51	183	206	180	6	8
51N406A	51	192	214	189	6	8
83N557A	83	332	235	210	8	8
83N562A	83	324	238	211	8	8
83N564A	83	328	236	214	8	8
123N829A	123	479	350	310	10	1 0
123N830A	123	477	353	303	10	10
123N836A	123	483	353	310	10	10

Chapter 8

Computational Study

In this chapter we present a computational study of all of the models introduced in previous chapters. As we use the heuristic to warm start the Branch-and-Price algorithm, we begin the computational study with a parameter calibration of the HGSDC in Section 8.1. Section 8.2 provides a comparison of the exact models that have been developed. The chapter is concluded in Section 8.3, with an in-depth examination of the performance of the HGSDC.

The arc-flow model is implemented in version 1.24.04 of the commercial software Xpress IVE, with version 3.10.0 of Xpress Mosel, and version 28.01.04 of Xpress Optimizer. The B&P algorithm and the genetic heuristic are coded in Java and run in version 2016.3.3 of IntelliJ IDEA, with version 1.8.0_111 of Java SDK. For the B&P, a library to Xpress IVE with the same versions and features as described above is added. All of the models are run on a computer with a 3.4 GHz Intel Core i7 processor and 32 GB of RAM, running Windows 10 Education.

8.1 Parameter calibration for the genetic algorithm

The performance of metaheuristics commonly depends on the values of several input parameters. This is especially true for evolutionary algorithms, such as the HGSADC (and our version), due to the amount of parameters. Vidal et al. (2012) perform an extensive parameter calibration for the initial implementation of the heuristic, using a meta-calibration approach to find the optimal values for the parameters. There is, of course, no parameter setting that can be viewed as optimal for all instances. However, instances with shared features may very well share ranges of the parameter values in

which the heuristic works well. Vidal et al. (2012) found that most of the parameters are independent from one another, although they were calibrated together. We assume that these independencies hold for our implementation as well, and so they are, with inspiration drawn from the work of Borthen and Loennechen (2016), mostly calibrated one by one. Nonetheless, all parameters subject to calibration were ranked according to their expected contribution to the quality of the solution.

8.1.1 The parameters

The parameters in the HGSDC that have been subject to calibration are listed in Table 8.1, alongside their value after calibration, and a brief description of their function. The parameters λ, μ, η^{Edu}, I^n and η^{Div} affect the search space and the number of iterations, and we therefore expect these to affect the performance most. Two parameters are omitted from the parameter calibration process, as finely adjusting them barely will influence the performance. These are discussed at the end of this section.

Table 8.1: Description of the parameters in the genetic algorithm that have been calibrated, and their associated value. Initial values are listed in parenthesis.

Parameter	Value	Description
μ	35 (25)	Minimum population size
λ	75 (75)	Number of offspring in each generation
I^n	1,000 (-)	Maximum number of iterations without improvement
η^{Div}	0.1 (0.1)	Proportion of iterations of improvement prior to diversification, such that $I^{Div} = \eta^{Div} \times I^n$
η^{Edu}	1 (1)	Proportion of offspring subject to education
η^{Elite}	0.3 (0.4)	Proportion of elite individuals, such that $n^{Elite} = \eta^{Elite} \times \mu$
η^{Close}	0.1 (0.2)	Proportion of close individuals evaluated for distance calibration, such that $n^{Close} = \eta^{Close} \times \mu$

8.1.2 The process of calibration

In order to calibrate the parameters, they were ranked according to their expected contribution to the makespan solution, prioritized in the order of their importance level. That is, the most important parameter's value is determined first, before the second parameter's value is determined, and so on. In each step, the final value of the parameter

is set. Although the underlying argument for this type of calibration is the independence of the parameters, we have concluded that four parameters should be calibrated pairwise, due to their close relations. The ranking of the parameters was found through preliminary testing, which is not conclusive evidence. Nevertheless, based on the independence discussed, we find that this approach to ranking is sufficient for this thesis. The ranking resulted in this sequence: η^{Edu}, I^n and η^{Div}, μ and λ, η^{Elite}, and η^{Close}.

The three instances in Test set 1 were used for calibration. Since the HGSDC is non-deterministic, each of the instances were run five times with each of the proposed parameter settings. For a given setting, the average run time and the average makespan for each instance is computed. As this is a relative calibration of parameters, the value of the makespan is not of importance to this discussion. However, the relative difference between the solutions provides a good indication of which parameter setting that is the better. Therefore, a gap (in terms of percentages) is computed between the average value and the best average value, with the following procedure: let a_{pi} be the average makespan value of parameter setting p for test instance i, and let a_i^{min} be the best average solution (with respect to the makespan) for all parameter settings for test instance i. The gap for instance i with parameter setting p, g_{pi}, is calculated as

$$g_{pi} = \frac{a_{pi} - a_i^{min}}{a_i^{min}} \times 100\%$$

Alongside the gap to the best average solution, the average computing time used by the heuristic given parameter setting p is included in the results. For the assessment of the parameter values, the gap is given priority over the computational times, as makespan is the objective. When a choice between two settings is to be made, the computation times may be taken into account. Anyhow, as these times are averages, they provide a good indication of how the computational effort is linked to the values given to the parameters.

8.1.3 The probability of education

The η^{Edu} parameter is the probability each offspring has of going through the education phase. The average gap and time for different values of η^{Edu} are provided in Table 8.2. For all three instances, best makespan solution is when education is performed on all offspring. That is, when $\eta^{Edu} = 1$. When $\eta^{Edu} = 0$, for all test instances, the heuristic yields average solutions having roughly twice the makespan value of the average

solution when the parameter has a value of 1. This is expected, as the algorithm has no mutation operator, which means that it is impossible to maintain diversity in the population when no education is initiated. For a parameter value of 0.5, the average gap is not that significant. However, we note that the average solution on the second largest test instance is close to zero. Interestingly, the average computation time is in this case more than doubled compared to the time for $\eta^{Edu} = 1$. This is contrary to the smallest and largest test instance, which have a larger gap for the average solution, but pairwise roughly equal computation time. We note that although education is a costly procedure, with respect to time, little doubt exists to whether all individuals should go through this phase.

Table 8.2: Results from running the HGSDC on Test set 1 for three values of η^{Edu}.

	η^{Edu}		
Instance	**0**	**0.5**	**1**
27N135A	82.7%-90	2.1%-1,014	**0**%-906
27N138A	109.8%-490	0.1%-4,116	**0**%-1,785
51N304A	111.2%-422	5.3%-1,628	**0**%-2,000

8.1.4 Non-improving iterations and diversification factor

The number of generations without improvement that are allowed to pass before the diversification process is initiated, I^{Div}, is given by a proportion of the maximum number of generations without improvement. That is, $I^{Div} = \eta^{Div} \times I^n$, where η^{Div} is the proportionality constant and I^n is the maximum number of consecutive non-improving generations. Due to this dependency, these are calibrated together. To get an indication on which values to test for I^n, the algorithm was run with $I^n = 5,000$ (a *very* high value) and $\eta^{Div} = 0.1$, several times. Data from one of the runs are plotted in Figure 8.1, which shows the number of generations, as intervals, before an improvement is found. The vast majority of the improvements are found within the first 100 generations, whereas almost none is found after 500 generations have surpassed. We note that the figure only shows one of several runs to measure this effect. However, the trend is very clear, yielding nearly the same results in all runs on different instance sizes. Between $1,900-5,000$ generations, no improvement was found in any of the runs, thus we conclude that there is no point letting I^n have a value of more than roughly 1,900. Testing multiple parameters in tandem is a time consuming procedure, quickly leading to many runs. For the

sake of the I^n parameter, which one would in general want to be as high as possible (as a lower parameter on average will be worse), we restrain ourselves to test for $I^n = 1,000$ and $I^n = 2,000$. It is clear that setting $I^n = 2,000$ seldom yields any individuals improving the solution. As $0.1 \times 5,000 = 500$, a forced diversification of the population is initiated every 500 iterations, given that no improvement is seen. The bar at $601-700$ generations in Figure 8.1 may indicate that the diversity measure is working, after some generations of mating and education. We note that the histogram shares commonalities with homogeneous Poisson processes (where the arrival rate of improvements is independent on the past). True or not, initiating a diversification phase can be viewed as "resetting" this process - although some burn-in time should be expected as the new individuals often will be poor in terms of makespan. For the parameter test, the average gaps and times for I^n and η^{Div} are shown in Table 8.3.

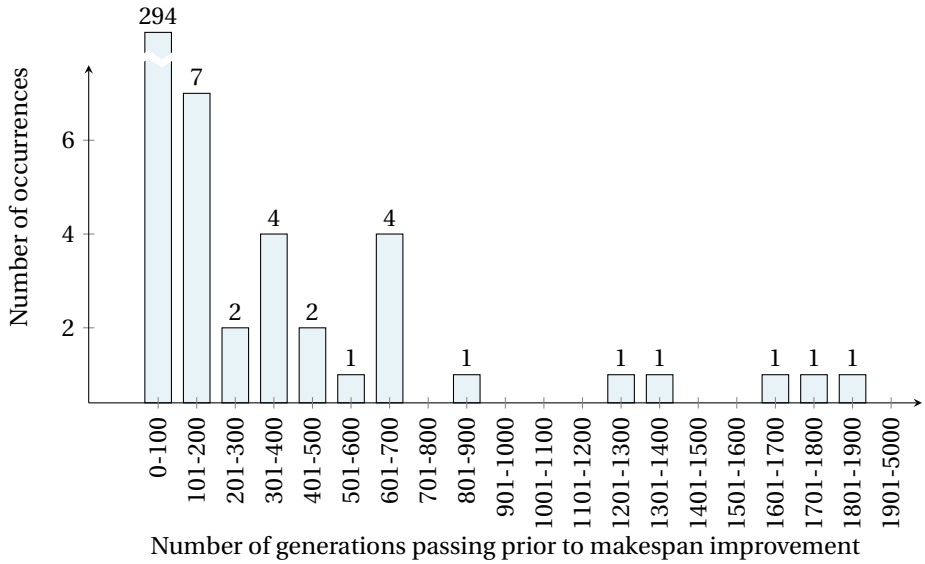

Figure 8.1: Histogram showing the number of generations prior to makespan improvement for an example run of the genetic algorithm.

The average computation time is in general considerably higher for $I^n = 2,000$ than for $I^n = 1,000$. This is not unexpected, as the search not only has a higher minimum number of iterations, but also because the search may find better solutions later than what is possible with $I^n = 1,000$. The best solution is found for $I^n = 1,000$ and $\eta^{DIV} = 0.1$. This means that the algorithm should diversify every 100 non-improving iteration. This is the same number of non-improving generations before diversifying as for $I^n =$

2,000 and $\eta^{DIV} = 0.05$. It is therefore surprising that this combination of parameters has a higher objective gap, as having a higher I^n implies having a higher number of total iterations. However, we note that the objective gap from the latter combination of parameters is under 1% for all test instances, which we view as insignificant. We conclude that increasing the number of I^n beyond 1,000 also increases the solution time, and has little effect on the objective gap. The parameters are therefore chosen to be $I^n = 1{,}000$ and $\eta^{DIV} = 0.1$.

Table 8.3: Results after running the HGSDC on Test set 1 for different values of I^n and η^{Div}.

		η^{Div}			
Instance	I^n	**0.05**	**0.1**	**0.4**	**0.7**
27N135A					
	1,000	2.4% - 799	**0**% - 1,097	2.4% - 1,682	2.7% - 978
	2,000	0.1% - 1,447	0.9% - 2,989	3.0% - 1,080	3.1% - 3,667
27N138A					
	1,000	2.1% - 1,955	**0**% - 2,081	2.1% - 2,663	0.6% - 2,256
	2,000	0.9% - 2,382	5.0% - 1,906	4.3% - 3,870	0.2% - 2,623
51N304A					
	1,000	1.7% - 2,578	**0**% - 1,912	0.1% - 2,640	7.8% - 2,008
	2,000	0.6% - 5,071	2.9% - 4,998	1.2% - 6,171	2.5% - 4,389

8.1.5 The population and generation size

As both the optimal population size and the generation size adjust the size of the population, they are to some degree dependent on one another, and we have chosen to perform a coordinated calibration of the two parameters. The objective gap between the average solutions, and the average running time for a set of predefined parameter values can be found in Table 8.4. We see that $\mu = 35$ and $\lambda = 75$ yield the best solutions for all of the instances in Test set 1, and they are therefore selected as the parameter values. We note that the average running times for the algorithm are very fluctuating, where the trend is increasing computation time with increasing λ. This should be expected, as a higher λ-value linearly increases the amount of education in each iteration - which on average is a significantly time consuming procedure.

Table 8.4: Results after running the HGSDC on Test set 1 for a range of values of μ and λ.

		μ			
Instance	λ	**15**	**25**	**35**	**50**
27N135A					
	25	3.6%-318	3.3%-306	3.6%-216	5.6%-178
	50	11.5%-173	2.5%-3,996	1.6%-539	0.1%-983
	75	4.2%-851	0.2%-930	**0**%-1,062	3.4%-553
	100	1.5%-979	0.1%-581	6.1%-948	2.1%-428
27N138A					
	25	9.0%-352	6.8%-429	8.9%-330	8.8%-266
	50	13.8%-1,008	4.1%-2,088	5.0%-675	10.4%-440
	75	14.8%-1,190	9.2%-1,331	**0**%-1,998	2.5%-1,368
	100	6.0%-1,056	5.9%-4,220	6.9%-1,967	4.2%-1,104
51N304A					
	25	13.2%-581	6.6%-693	12.3%-813	8.2%-1,110
	50	13.3%-1,371	7.6%-1,397	4.8%-2,693	4.3%-2,315
	75	10.2%-1,860	6.7%-2,388	**0**%-2,238	9.5%-2,451
	100	10.3%-2,468	14.7%-2,568	2.7%-2,300	4.6%-2,340

8.1.6 The proportion of elite individuals

Table 8.5: Results when running the HGSDC on instances in Test set 1 for different values of η^{Elite}.

	η^{Elite}				
Instance	**0.2**	**0.3**	**0.4**	**0.5**	**0.6**
27N135A	2.4%-640	**0**%-587	1.8%-942	7.0%-497	1.3%-915
27N138A	0.2%-1,432	**0**%-1,965	1.7%-1,835	3.9%-4,560	1.7%-1,786
51N304A	6.4%-1,582	**0**%-1,816	3.2%-2,188	3.4%-2,018	1.3%-2,324

The η^{Elite} parameter is a measure of the importance of regular fitness versus the importance of biased fitness. A higher parameter value signals that regular fitness should be given a higher weight than biased fitness. Table 8.5 lists the results from the computations with discrete values of η^{Elite} between 0.2 and 0.6. For all of the instances, the best average solution is obtained with $\eta^{Elite} = 0.3$, seemingly increasing when adjusted in any direction. Again, the average running times are to some extent fluctuat-

ing, in general increasing for larger test instances. For the two largest instances, the best parameter value with respect to the average makespan value, also yields the lowest computation time. Due to the general fluctuation of the computation times, we view this as insignificant, although noting that on average, the computation time is low for $\eta^{Elite} = 0.3$. All in all, there is no doubt that these results indicate that η^{Elite} should be given a value of 0.3.

8.1.7 The neighborhood factor

Table 8.6: Results when running the HGSDC on Test set 1 for different values of η^{Close}.

	η^{Close}				
Instance	**0.1**	**0.2**	**0.3**	**0.4**	**0.5**
27N135A	**0**%-1,194	4.8%-1,252	8.8%-1,300	7.9%-1,289	3.1%-1,231
27N138A	1.5%-1,815	**0**%-1,789	13.5%-2,031	6.9%-1,912	3.3%-1,848
51N304A	**0**%-2,283	6.9%-2,441	4.3%-2,382	3.7%-2,367	8.5%-2,477

The fraction of the closest neighbors each individual is compared with, when assessing to which degree the individual is contributing to the diversity of the population, is described by the parameter η^{Close}. A lower η^{Close} increases the importance of the individual having a large distance to its nearest neighbors relative to the importance of being different to distant neighbors. The average gap and time for different values of η^{Close} from the calibration runs, can be found in Table 8.6. Seen from the table, $\eta^{Close} = 0.1$ yields the best solutions for two of the three instances, while $\eta^{Close} = 0.2$ yields the best solutions for one instance. We view the increase in the gaps of the average makespan solutions when increasing the parameter value as significant. The average computation time, however, is about equal for all of the tested parameter values. Little doubt exists to whether this parameter should be low, which is why $\eta^{Close} = 0.1$ is chosen for the HGSDC.

8.1.8 Parameters not subject to calibration

All but two parameters have been mentioned in the previous sections, as they have been omitted from the calibration altogether. The maximum runtime is a constant which indicates at what time the HGSDC shall terminate, if it has not already terminated. This time limit was never reached during the calibration phase, and is not expected to be

reached at any point running the heuristic in instances generated for this thesis. Note that, for the sake of the results in this thesis, we do not want the heuristic to reach this upper time limit. Therefore, if the test instances get sufficiently large, the maximum runtime parameter should be given a higher value.

The problem of prohibiting U-turns has been solved by adding a cost (in terms of time units) to the individuals that include U-turns. In the case of the MFARPOS, where we want to disallow all U-turns completely, a sufficiently low cost is equivalent to allowing infeasible solutions. However, when increasing the cost of U-turns by multiple orders of magnitude, the probability for U-turns to survive in the population is so low that it is practically equivalent to removing them. Therefore, the static value of 1,000 time units is added to the makespan for each U-turn in the larger vehicles' routes. Making this value larger may influence the burn-in time (the time it takes to remove all solutions with forbidden turns). However, in a preliminary test, with a U-turn cost of 1,000, all solutions with U-turns ended up having a lower fitness value than those without U-turns. Note that putting a price on U-turns only is considered in the final section of the computational study.

8.2 Exact solution approaches

In this section we examine how the different exact solution methods perform. Three variations of the B&P algorithm are considered, where the only difference is the starting point of the column generation in the root node. The *cold start* model means that the column generation starts without any preprocessing, or columns added. When referring to *warm start without columns*, the basic version of the HGSDC heuristic is initially run to find a solution to the problem, thus tightening the value of T^{Max} in the algorithm - decreasing the maximum time of a label. *Warm start with columns* uses both the upper bound for the makespan, and all of the paths (columns) generated by the heuristic whose time are shorter than or equal to the heuristic solution. The B&P algorithm with the best column generation approach is further compared to the MIP model developed prior to this thesis, in order to evaluate the significance of the method. Test set 2, 3, and 4 are used for the exact models. A time limit of 36,000 seconds was set for all of the runs. Note that for the warm start approaches, the computation time of the heuristic is included in the total time. It is, however, only for the smaller instances that this time is relatively significant.

8.2.1 Comparing the column generation approaches

The three models were initially run on Test set 2. The computational results from these runs can be found in Tables A.1, A.2, and A.3 in the appendix. A summary is provided in Table 8.7. The *Total time* column shows the total time of running the algorithm before termination. *IP gap* measures the percentage gap from the best integer solution found within the respective algorithm, to the best known dual bound. That is, *IP gap* $= \frac{z^{IP}-z^*}{z^*}$, where z^{IP} is the best integer solution found by the respective algorithm, and z^* is the best known dual bound for the instance, rounded up to the nearest integer. Note that for all instances except 16N25A, the dual bound is optimal. *Lower gap* is the percentage gap from the dual bound obtained in the model to the best known dual bound; as calculated by *Lower gap* $= \frac{z^*-z^{LB}}{z^*}$, where z^{LB} is the best dual bound obtained by the model, rounded up to the nearest integer. *Root gap* refers to the percentage gap from the root node solution of the model to the best known dual bound. It is given by *Root gap* $= \frac{z^*-z^{RN}}{z^*}$, where z^{RN} is the root node solution, rounded up to the nearest integer.

Table 8.7: Summary of the computational results from running the B&P algorithms on Test set 2. The asterisk indicates that 16N52A is omitted from the averages, as some models failed to find a root node solution for this instance within the time limit. *n/a* indicates that the algorithm did not manage to find the corresponding result within the time limit. The best average values are marked in bold.

	Cold start				**Warm start without columns**				**Warm start with columns**			
Instance	**Total time(s)**	**IP gap(%)**	**Lower gap(%)**	**Root gap(%)**	**Total time(s)**	**IP gap(%)**	**Lower gap(%)**	**Root gap(%)**	**Total time(s)**	**IP gap(%)**	**Lower gap(%)**	**Root gap(%)**
8N21A	2	0	0	0	44	0	0	0	43	0	0	0
8N28A	9	0	0	0	590	0	0	0	430	0	0	0
9N27A	162	0	0	1.25	269	0	0	1.25	149	0	0	1.25
10N38A	1,211	0	0	0	501	0	0	0	2,845	0	0	0
11N33A	2,292	0	0	10.42	819	0	0	6.25	761	0	0	6.25
12N37A	>36,000	8.11	1.80	6.31	12,317	0	0	0	>36,000	5.41	0	0
13N46A	>36,000	6.67	2.22	2.22	>36,000	4.44	0	0	>36,000	5.56	0	0
14N46A	>36,000	29.03	5.38	5.38	>36,000	5.38	1.08	1.08	>36,000	4.30	1.08	1.08
15N44A	>36,000	8.04	18.75	18.75	>36,000	0	3.57	4.46	>36,000	0	3.57	4.46
16N52A	>36,000	n/a	n/a	n/a	>36,000	0.76	n/a	n/a	>36,000	0.76	2.27	2.27
Average*	>16,408	5.76	3.13	4.92	**>13,616**	**1.09**	**0.52**	**1.45**	>16,000	1.70	**0.52**	**1.45**

A general observation in all of the models, is that the majority of the computation time is spent in the subproblems. This is clear from Tables A.1, A.2, and A.3. Our interpretation of this results is that the dominance criteria is not working as well as anticipated, such that a vast amount of labels are generated and evaluated in the subproblems. This is a problem in order to prove that the current solution of the RMP is optimal (within the given conditions). In order to do that, the subproblems have to iterate through all of the undominated labels, and prove that none of them have a negative reduced cost, all of which is very time consuming. When comparing the time each model uses in the subproblems, it is clear that tightening the value of T^{Max} by including an upper bound on the makespan will cut the time spent in the subproblems. The time resource for the labels is tighter in those cases, which reduces the amount of feasible labels. Indeed this also underlines the value of the heuristic solutions. Also, as should be expected, a tighter T^{Max} provides a better root node solution, as it restricts the length of the routes (in terms of time units), and thereby restricts the solution space for the LP relaxation.

When warm starting the columns generation, the B&B tree grows with a smaller rate than that of the cold start approach, and the total number of columns generated is correspondingly smaller (roughly half as many as the cold start). It is clear that the number of nodes generally increases with increasing instance size. However, when comparing the warm start approaches, the number of nodes with respect to the instance size seems more fluctuating when including the columns from the heuristic. Thus, the time used to prove optimality in the nodes varies significantly from instance to instance. In general, the tree may become large on instances where the algorithm fails to reach, and prove, optimality. In order for the tree to grow, there must exist columns with fractional value in the optimal solution of the upper nodes. Thus, it would seem that further work can be put into the branching strategy to lessen the number of nodes in the tree.

For the larger instances, a significant portion of the time is spent trying to find new integer solutions with the known columns. Although being time consuming, the solutions are seemingly good, with a larger bound of 5.56% from the best known dual solution.

Throughout all models, it is clear that as the size of the test instance increases, although marginally, the computational effort increases significantly. However, all versions are able to solve and prove optimality in the first half of the test set, where the smallest instances are solved within a fraction of the time limit. We note that instance 11N33A may be viewed as an outlier among the instances. For all models, the root node

solution for this instance is very poor (ranges from 6.25% to 10.42% gap to the optimal value); and for the warm start approaches, the heuristic solution is weak (8.33% gap).

The computational results indicate that warm starting the column generation, both with and without columns, significantly improves the Branch-and-Price algorithm. It is able to find better integer solutions on the larger half of the test set, compared to the cold start approach. On the other hand, it is hard to argue that adding the columns provided by the genetic algorithm to the B&P algorithm is very valuable. This method is able to find the best integer solution in one instance, in addition to solving the root node of the largest instance within the time limit. When taking the average over all instances, the better model is that which starts only with an upper bound on the makespan value. Additionally, this is on average the best model to find integer solutions to the test set, with an average gap of 1.06%, compared to the second warm start approach, which has a gap of 1.60% (note that these numbers are not included in the table as they are averages over all ten test instances). As solutions with the best makespan objective are the primary goal of our study, we continue with this version of the column generation in the B&P algorithm for the remaining sections in the computational study of exact models.

8.2.2 Comparing the path-flow model and the arc-flow model

In Table 8.8, we present results from the implementation of the arc-flow model, alongside the obtained results from the B&P algorithm with the warm started column generation (with no columns added), when solving the instances in Test set 2. As seen from the computational results, the arc-flow model proved optimality for seven test instances, whereas the path-flow model proved optimal solution for six (it found the optimal integer value in test instance 15N44A as well, but this was not proved within the time limit).

Comparing the gap of the root node solutions for the two different models, the average value for the path-flow formulation (1.53%) is significantly closer to the best known dual solution than the arc-flow model (5.95%). Barnhart et al. (1998) point out that such a reformulation, where the number of constraints in the master problem is fewer, although with increasing number of variables, may yield tighter LP relaxations. We therefore expected this effect in our problem as well.

On the larger test instances in Test set 2, the arc-flow model outperforms the path-flow model, when it comes to finding the optimal solution within the time limit. The arc-flow model proved optimality in three of the four largest instances, with an IP gap

Table 8.8: Key results from running the arc-flow model and the path-flow model on Test set 2. Asterisk indicate that the associated value is taken from the warm start with columns (as they would be equal) in order to get a better basis of comparison of the root node solutions.

	Arc-flow model			Path-flow model		
Instance	**Total time(s)**	**IP gap(%)**	**Root gap(%)**	**Total time(s)**	**IP gap(%)**	**Root gap(%)**
8N21A	2	0	0	44	0	0
8N28A	>36,000	1.15	0	590	0	0
9N27A	26	0	1.25	269	0	1.25
10N38A	>36,000	3.90	0	501	0	0
11N33A	66	0	10.42	819	0	6.25
12N37A	93	0	6.31	12,317	0	0
13N46A	5,538	0	2.22	>36,000	4.44	0
14N46A	24,590	0	5.38	>36,000	5.38	1.08
15N44A	3,257	0	18.75	>36,000	0	4.46
16N52A	>36,000	0.76	15.15	>36,000	0.76	2.27*
Average	>14,157	0.58	5.95	>15,854	1.06	1.53

of 0.76% in the largest, whereas the path-flow failed to prove optimality in all of them (and not even solved the root node in 16N52A). As discussed in the previous section, the computation time for the B&P algorithm increases rapidly when the number of nodes in the graph increases.

The models are tested on yet another set of larger instances, namely Test set 3. The results are summarized in Table 8.9. *Best IP sol* is the makespan value of the test set when solved by the respective model, and the numbers inside the brackets show the heuristic solutions for the path-flow model. *Root sol* is the root node solution rounded up to the nearest integer. The path-flow model is unable to solve the root node to optimality in all of the instances, and thereby improve its heuristic solution. The arc-flow model, on the contrary, manage to find an integer solution on the two smallest of the five instances, although the IP solution on test instance 22N108A is very poor, with 46% gap to the heuristic solution. We conclude, with special emphasis on the path-flow approach, that these models are very weak for instances above roughly 20 nodes and 100 arcs.

A final observation from Table 8.8 is that the arc-flow model is unable to prove optimality in test instance 8N28A and 10N38A, two of the instances with the lowest number

Table 8.9: Key results from running the best warm started column generation model and the arc-flow model on Test set 3. Numbers in brackets show the heuristic solution for the path-flow model. *n/a* indicates that the algorithm did not manage to find the corresponding result within the time limit.

	Arc-flow model			Path-flow model		
Instance	**Total time (s)**	**Best IP sol**	**Root sol**	**Total time (s)**	**Best IP sol**	**Root sol**
22N99A	>36,000	230	187	>36,000	233 (233)	n/a
22N108A	>36,000	312	173	>36,000	213 (213)	n/a
24N123A	>36,000	n/a	174	>36,000	202 (202)	n/a
26N116A	>36,000	n/a	188	>36,000	246 (246)	n/a
27N139A	>36,000	n/a	162	>36,000	251 (251)	n/a

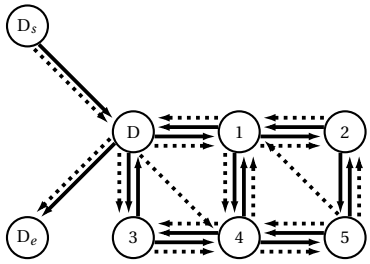

Figure 8.2: Illustration of the graph describing the road network in test instance 8N28A. Solid and dotted arrows represent lanes and sidewalks, respectively.

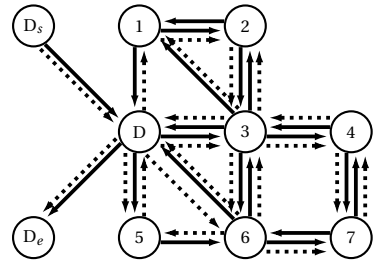

Figure 8.3: Illustration of the graph describing the road network in test instance 10N38A. Solid and dotted arrows represent lanes and sidewalks, respectively.

of nodes, within the time limit of 36,000 seconds. This is so, even if the gap from the root node solution to the optimal solution is 0%. The road networks for these instances are presented in Figures 8.2 and 8.3. As one can clearly see in the graphical representations, these graphs are very dense, with synchronization constraints associated with almost every lane. As the path-flow model solves these instances to optimality within 600 seconds, this may be an indication that there is a significant difference on the quality of the models on such graphs. We therefore continue our comparison study on Test set 4, which comprises instances where the road networks are small and dense, with many synchronization constraints.

Table 8.10 provides a summary of the results from running the arc-flow model and the path-flow model on Test set 4. *Best IP sol (%)* refers to the percentage gap between the best integer solutions obtained with the models. It is calculated according

to *Best IP sol (%)* = $\frac{z^{IP}_{AF} - z^{IP}_{PF}}{z^{IP}_{PF}}$, where z^{IP}_{AF} and z^{IP}_{PF} are the best integer solutions of the arc-flow and path-flow model, respectively. *Root sol (%)* is the percentage gap between the root node solutions in both models, and is calculated as *Root sol (%)* = $\frac{z^{R}_{PF} - z^{R}_{AF}}{z^{R}_{PF}}$, where z^{R}_{AF} and z^{R}_{PF} are the rounded root node solutions of the arc-flow and path-flow model, respectively. A positive value of *Best IP sol (%)* and *Root sol (%)* indicate that the path-flow model found the lowest integer solution, and the tightest root node solution, respectively.

Inspecting the results clearly reveal that the implementation of the arc-flow formulation does not perform well, relative to the path-flow model. In all instances included in the set, the best integer solution is given by the path-flow model, leaving the average gap of the best integer solution at 13.35%. As the gap for all instances is > 5%, which is a lower bound for the gap to the optimal solution, this implies that the arc-flow model in all instances did not even come close to finding the optimal solution. The root node solution, on the contrary, is insignificantly (< 1% difference on average) in favor of the path-flow model. We also note that for all but two instances, the path-flow formulation was able to improve the makespan solution provided by the heuristic. In instance 11N60A, this improvement was 15%, indicating not only that the heuristic provided a not-so-good solution, but also that the path-flow model's ability to perform such an improvement even when the value of T^{Max} is far from the optimal makespan solution.

Table 8.10: Results from running the arc-flow model and the path-flow model on Test set 4. Numbers in brackets show the heuristic solution for the path-flow model.

	Arc-flow model			**Path-flow model**			**Difference**	
Instance	**Total time(s)**	**Best IP sol**	**Root sol**	**Total time(s)**	**Best IP sol**	**Root sol**	**Best IP sol(%)**	**Root sol(%)**
11N52A	>36,000	94	80	>36,000	88 (91)	80	6.82	0
11N54A	>36,000	99	82	>36,000	94 (94)	83	5.32	1.20
11N55A	>36,000	104	84	>36,000	96 (96)	84	8.33	0
11N56A	>36,000	91	66	>36,000	76 (79)	67	19.74	1.49
11N59A	>36,000	91	64	>36,000	85 (86)	66	7.06	3.03
11N60A	>36,000	93	63	>36,000	70 (82)	63	32.86	0
Average							13.35	0.95

We motivate the difference of the quality of the models by the following reasoning. In the decomposed model (path-flow model), the routes in the SPs are made with no concern to the synchronization criteria, as the SPs only get dual values from the syn-

chronization constraints in the RMP. In effect, these dual values make the SPs prioritize some arcs above others. This stands in clear contrast to the arc-flow model, where synchronization constraints are complicating the structure of the formulation, and must be continuously accounted for when the routes are being constructed.

8.2.3 Summarizing the results for the exact models

To summarize the computational study of the exact models, we conclude from the results on Test set 2 that the path-flow model yield better root node solutions than the arc-flow model on small instances. For the medium sized instances in Test set 3 the path-flow model failed to reach a root node solution within the time limit (and the arc-flow model fail to reach an IP solution in 3 of 5 instances). Thus, it is clear that the size of the instances is in the limit of what these models can handle. As seen from the comparison of the different column generation approaches for the B&P algorithm, the value of T^{Max} influences the quality of the root node solution significantly. Therefore, it can be the case that the maximum parameter values simply were not tight enough when running Test set 4, making the observed difference in the root solution biased towards the MIP's.

The perhaps most interesting result from this comparison is the fact that the warm started column generation approach is paramount to the arc-flow model on instances with small and dense road networks with many synchronization constraints. Thus, we can conclude that the decomposed formulation is very sensitive to the number of nodes in the instance graph, whereas the compact formulation is more sensitive to the number of arcs (often associated with many synchronization relations). However, in order to find good solutions for MFARPOS on realistic instances, heuristic solution methods are needed.

8.3 Evaluating the genetic algorithm

The rest of the computational study is concerned with an evaluation of the HGSDC. The *Basic HGSDC* refers to the version of the heuristic that has been used in all previous analysis (as the pre-solver for the warm start approaches). We emphasize that this model does not have adaptive diversity control, as the framework presented by Vidal et al. (2012), which adaptively changed the number of infeasible solutions allowed in the population, do not apply for the MFARPOS. *HGSDC with adaptiveness* refers to the genetic algorithm including the adaptiveness measure we proposed in Chapter 6.2.1 for this specific problem. Initial findings, which are the results obtained when using the Basic HGSDC to warm start the column generation models, are promising. In this section, we evaluate the performance of the HGSDC on these instances, now with ten runs on each instance in Test set 2. The heuristic is further tested on the largest instances generated, namely Test set 5. The stability of the heuristic and U-turn constraints are also considered in this section.

8.3.1 Testing the heuristics on small to medium sized instances

For each instance in Test set 2, both the Basic HGSDC and the HGSDC with adaptiveness were run a total of ten times each. Key results are presented in Table 8.11. As a measure of stability, the *relative standard deviation* (RSD) is used. That is, $RSD = \frac{S_x}{\bar{x}}$, where S_x is the sample standard deviation given by $S_x = \sqrt{\frac{\sum(x_i-\bar{x})^2}{N-1}}$, where N is the sample size (10), and $\bar{x}$ is the sample mean, which is given by $\bar{x} = \frac{\sum x_i}{N}$. The sigmas indicates that summations include all of the makespan values, given by x_i, where $i = \{1,\dots,N\}$. In the table, the RSD is multiplied by 100, thus providing the size of the standard deviation compared to the mean value, in terms of percentages. The *Gap*-column represents the percentage deviation to the best known dual solution of the instance, and is calculated by the same formula as the *IP gap* for the exact models. In the *Average*-columns, the gap refers to the gap of the average solution to the best dual bound known. The maximum run time for this test set is 6,000 seconds.

Table 8.11: Summary of the computational results from running the Basic HGSDC and HGSDC with adaptiveness on Test set 2. The best average values are marked in bold.

	Basic HGSDC					**HGSDC with adaptiveness**				
	Average		**Best**			**Average**		**Best**		
Instance	**Gap(%)**	**Time(s)**	**Gap(%)**	**Time(s)**	**RSD** $\times 100$	**Gap(%)**	**Time(s)**	**Gap(%)**	**Time(s)**	**RSD** $\times 100$
8N21A	0	33	0	33	0	0	60	0	60	0
8N28A	1.15	447	0	441	1.69	3.68	243	0	297	2.57
9N72A	3.50	376	0	504	3.54	3.37	220	0	234	3.61
10N38A	8.57	445	5.19	780	3.20	10.91	307	5.19	654	2.98
11N33A	4.69	267	0	473	5.14	6.67	186	0	238	3.94
12N37A	6.67	310	0	659	6.32	8.11	231	0	351	5.10
13N46A	9.89	1,057	4.44	1,221	3.91	13.33	513	2.22	1,047	4.09
14N46A	7.42	1,020	5.38	1,196	1.10	8.82	654	1.08	964	3.99
15N44A	5.71	642	3.57	986	1.89	8.21	406	3.57	645	3.19
16N52A	3.91	630	0.76	886	3.10	10.91	538	4.55	842	4.18
Average	**5.15**	523	1.93	718	**2.98**	7.40	336	**1.66**	533	3.36

Five of the instances in Test set 2 were solved to optimality with both versions of the heuristic. The smallest instance, 8N21A, was solved to optimality in all 20 runs. Thus, we conclude that below a certain limit, the search space is small enough to find the optimal solution, whatever the starting point of the heuristic. For the basic version of the heuristic, the gaps of the average solutions to the best known dual bounds are on average 5.15%, while the average gap of the best solutions within each of the ten respective runs, is 1.97%. For the adaptive version, the average gap of the average solutions is higher, with a value of 7.40%, while the average gap of the best solutions is lower, with a value of 1.66%. The difference may be due to the low amount of time spent diversifying for the version that adaptively changes the n^{Elite} parameter. If n^{Elite} increases too fast, the search in many runs may converge to a "bad" local optimum. The average value can therefore be worse than if diversification was valued more throughout the generations, which would lead to more exploration of the solution space. On the off-chances where the region of convergence is near the optimal solution, having a high n^{Elite} is better. This increases the importance of elitism when an increasing number of generations have passed, and focuses the search deeper into this region of the search space by allowing more solutions with similarities, thus leading to better solutions. Such a statement should of course be backed by a statistical analysis, or a deeper inspection of the behavior of the heuristic. However, we view this result as quite weak, and leave it with the above comments.

When inspecting elements in Table 8.11 further, one can clearly see a trend in the gaps of the average solution in each instance. For all but instance 9N72A, the gap of the average solution is smaller for the Basic HGSDC heuristic. As the same denominator is used to calculate the gaps for both versions of the heuristic, this result can only be true if the average values are equally better for the Basic HGSDC. A simple statistical analysis can be carried out on this observation. Let the null hypothesis, H_0, be that the mean (average) values should be equal. We can now compute the p-value that eight or more out of the ten instances would yield a better mean in favor of the Basic HGSDC (we use eight out of ten, not eight out of nine to favor the uncertainty). If H_0 is true, then there should in all instances be a maximum 50% chance that the best average gap should be for any of the two heuristics (since we do not know the probability of the means being equal). Note that this holds no matter what the respective distributions or standard deviations are. As the instances are independent, then in the limit, the outcomes follow a binomial distribution with a probability of 0.5 of the Basic HGSDC having the smallest

mean. Thus, a strict upper limit for the p-value can be computed as

$$\sum_{k=8}^{10} \frac{10!}{(10-k)!k!} 0.5^k (1-0.5)^{10-k} = 0.0547$$

This indicates that H_0 would not been rejected in a 95% confidence interval. However, we note there is a high probability that the Basic HGSDC is in fact better on average.

With respect to the increase in the gaps when increasing the graph size of the network, it is reasonable that this should happen - although unclear to what extent. We now investigate whether the spread, when scaled by the mean, increases when increasing the size of the road network. Four linear regressions have therefore been carried out. Figure 8.4 is the linear regression plot of the RSD×100 as a function of nodes for the Basic HGSDC, where Figure 8.5 is the linear regression plot of the RSD×100 as a function of arcs in the graph. Figure 8.6 and Figure 8.7 show similar regressions for the HGSDC with adaptiveness. In all cases, the regression is carried out by minimizing the sum of the squared errors. By mere inspection, it is clear that the RSDs are more scattered for the Basic HGSDC, making the regression line more sensitive to the outliers. In fact, the slope of the regression line is negative in Figure 8.4 and Figure 8.5 if one removes the data point corresponding to instance 8N21A. This is not the case for the HGSDC with adaptiveness. For these plots, we simply note that this is so, as our analysis show that the data available is not enough to show a statistical significance either way. This is mainly due to the fact that although the RSD is on average very close for the two heuristics, the data points for the Basic HGSDC deviate more from the regression line.

Another interesting observation from Table 8.11 is recognizing that the computation time for all of the best makespan solutions, is longer than the average computation time. This is true in nine out of ten instances for the basic version, and in all instances for the version with adaptiveness. This may be an indication that the solution space for this problem is complex, and that improvements to find the best solution generally may come after many generations, or iterations without improvement, and thus increasing the computation time.

With respect to the fleet sizes in the instances, we note that both versions are able to solve all instances with five or less vehicles to optimality. None of the instances with six vehicles (which is the maximum for this set) was solved to optimality. When there are nearly as many synchronization relations as there are lanes, and fewer smaller vehicles than plowing trucks, the importance of finding a good route for the plowing trucks is

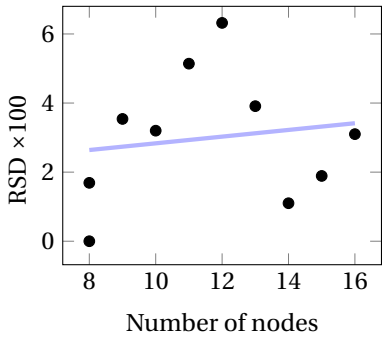

Figure 8.4: Regression plot of the relative standard deviation as a function of the number of nodes for the Basic HGSDC. Black dots indicate data points, whereas the blue line is the regression line.

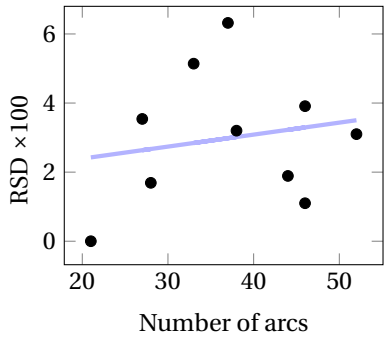

Figure 8.5: Regression plot of the relative standard deviation as a function of the number of arcs for the Basic HGSDC. Black dots indicate data points, whereas the blue line is the regression line.

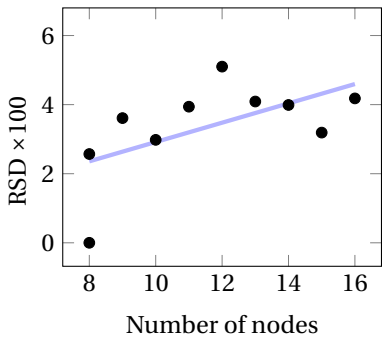

Figure 8.6: Regression plot of the relative standard deviation as a function of the number of nodes for the HGSDC with adaptiveness. Black dots indicate data points, whereas the blue line is the regression line.

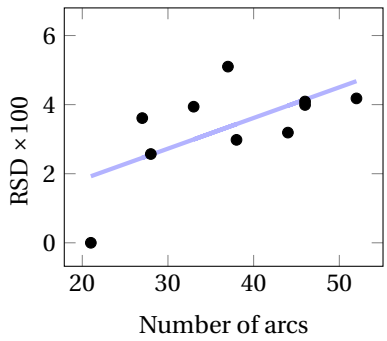

Figure 8.7: Regression plot of the relative standard deviation as a function of the number of arcs for the HGSDC with adaptiveness. Black dots indicate data points, whereas the blue line is the regression line.

less critical. As there are both more plowing trucks, and the time used to plow a lane is less than the corresponding sidewalk, many routes for the plowing trucks are very good and may even be candidates for optimal solutions. In effect, this should make the search primarily a search for good routes for the smaller vehicles. When the number of smaller vehicles is the same as the number of larger vehicles, the probability that the plowing trucks set the makespan increases. This, in turn, increases the importance

of finding good routes for vehicles of both fleets, decreasing the number of optimal solutions, and most likely the probability of finding one of them. As there are three vehicles in both fleets in all of the five instances that were not solved to optimality by the heuristic, this effect may be significant.

8.3.2 Testing the heuristics on large instances

In this section follows an evaluation of the two version of the heuristic when solving the instances in Test set 5, a set containing medium to large instances. Due to the size of the graphs, no lower bound was possible to obtain for any of these instances. Thus, the following discussion is primarily centered around the solutions (in terms of makespan value), the time to obtain the solutions, and the measure of stability as introduced in the previous section. All of the instances were solved a total of five times per model. The key results obtained from running the two versions of the heuristic on Test set 5 are summarized in Table 8.12. The *Solution* column indicates the solution in terms of makespan, whereas the $RSD \times 100$ is again the measure used to analyze the spread of the solutions. To make sure the searches would be capable of terminating within the maximum run time limit, the maximum run time on these instances was further increased to $10,000$ seconds.

It is clear that on the larger instances, the HGSDC with adaptiveness perform just as well as the Basic model. It has the best makespan solution and the best average solution in five out of the nine instances, with seemingly no preference regarding instance size. In contrast to the findings in the previous section, the spread of the solutions are in general better for the adaptive version of the heuristic on these instances. This is especially noticeable in the runs of instance 51N373A, in which the solutions have a remarkably low spread value of 2.33%. This stands in great contrast to the basic version, where the sample standard deviations is 11.69% of the sample mean. The heuristic with adaptiveness has the lowest relative spread in six out of the nine instance, often being much lower than its counterpart. It is tempting to carry out a statistical analysis to test the hypothesis that the spread is different for the two models. This should be performed with a χ-squared test. However, we would have to assume that the makespan solutions are normally distributed, which is not feasible to verify with only five data points. Additionally, the associated distribution of the test statistic will have too few degrees of freedom to yield any significant results.

Table 8.12: Summary of the computational results from running the Basic HGSDC and HGSDC with adaptiveness on Test set 5. The asterisk marks that the run terminated as result of surpassing the maximum run time. We note that this happened only on the respective run of the heuristic.

	Basic HGSDC					**HGSDC with adaptiveness**				
	Average		**Best**			**Average**		**Best**		
Instance	**Solution**	**Time(s)**	**Solution**	**Time(s)**	**RSD** ×100	**Solution**	**Time(s)**	**Solution**	**Time(s)**	**RSD** ×100
51N373A	523	2,066	437	2,596	11.69	454	3,676	440	2,308	2.33
51N389A	494	3,079	440	3,305	12.89	485	4,517	438	9,298	6.94
51N406A	519	3,553	460	6,995	10.00	524	2,929	492	2,086	7.59
83N557A	710	1,806	625	829	10.07	646	4,616	589	7,768	7.62
83N562A	712	1,532	674	1,256	5.53	774	3,937	668	4,637	9.69
83N564A	682	1,714	627	3,513	8.14	647	4,358	593	4,133	8.22
123N829A	947	2,856	810	4,612	12.64	960	5,317	877	10,000*	6.63
123N830A	915	2,091	779	1,621	15.95	920	2,818	801	4,544	8.52
123N836A	1,125	1,576	980	1,623	7.26	1,025	2,692	910	3,968	9.30
Average		2,253		2,928	10.46		3,873		5,416	7.43

The solution development of the heuristic solutions for instance 51N373A is plotted in Figure 8.8 and Figure 8.9, for the Basic HGSDC and the HGSDC with adaptiveness, respectively. Each line in the plots shows the development of the makespan value during one run of the heuristic. The plots indicate that the heuristic with adaptiveness has on average a steeper slope in the first few hundred seconds, indicating that valuating elitism low initially, is a good idea. Additionally, all but two of the runs with the adaptive version of the heuristic last for quite a long time. The ones terminating first corresponds to the ones with the best solution at the termination point. This is contrary to the basic HGSDC, where the first run terminating (the orange line in Figure 8.8) also is the run with the worst return value of the makespan at the termination run time. By graphing the results from other instances, the development of the makespan shared these properties more often than not.

On these larger instances, we find no clear correlation between the size of the graphs in the instances and the computation time. In fact, the average time used to solve the smallest instance (51N373A) is greater than the time to solve the largest (123N836), for both versions of the heuristic, even though the latter contains more than twice the number of both nodes and arcs than the former. When comparing the solution methods, the HGSDC with adaptiveness has the longest average run time in eight out of the nine instances. This is an important notice, since long computation time often is associated with better solutions. In seven out of the nine instances in Test set 5, the best solution was also associated with the longest computation time, with respect to the version of the heuristic. To clarify, consider instance 123N836A. The HGSDC with adaptiveness obtained the best makespan solution of 910 with a computation time of 3,968 seconds, whereas the Basic HGSDC obtained its best solution (980) in 1,623 seconds, which is significantly lower.

Figure 8.10 illustrates the corresponding n^{Elite} value for the runs of the HGSDC with adaptiveness on instance 51N373A. Note that the horizontal axis in this figure is the number of generations since initialization, not the computing time. There is no guarantee that varying the n^{Elite} parameter yield better solutions, but we note that for several instances, the n^{Elite} value reached above 20 before an improvement in the makespan occurred. This correspond to an eliteness percentage of at least $\frac{20}{35} = 0.6$, which means that at least 60% of the population will survive due to makespan value - with no regard to their contribution of diversity. We conclude that in some instances, a higher parameter value, and thus a more intensified search, may be positive with respect to improving the makespan.

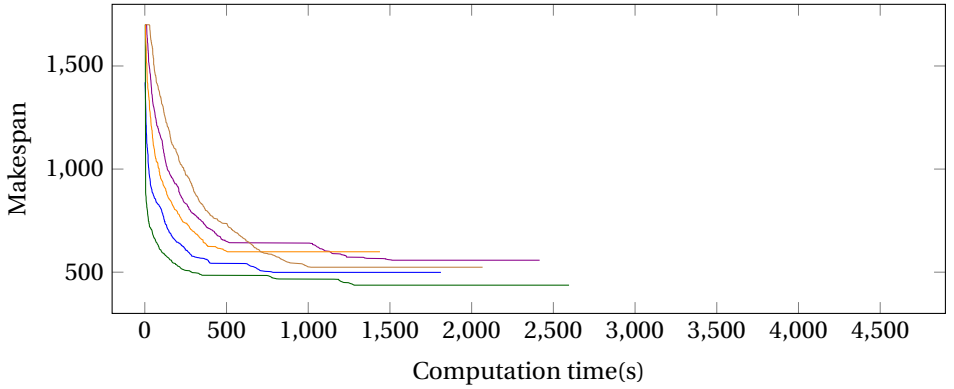

Figure 8.8: The objective value of the best solution in the population as a function of computing time in the five runs on 51N373A for the basic HGSDC.

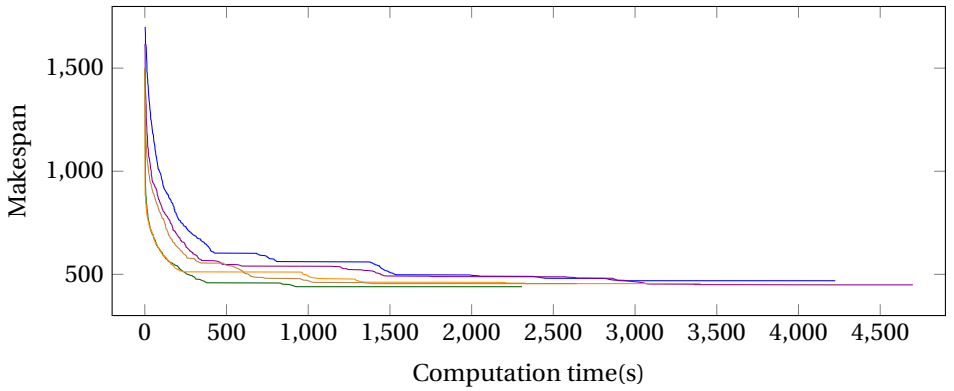

Figure 8.9: The objective value of the best solution in the population as a function of computing time in the five runs on 51N373A the HGSDC with adaptiveness.

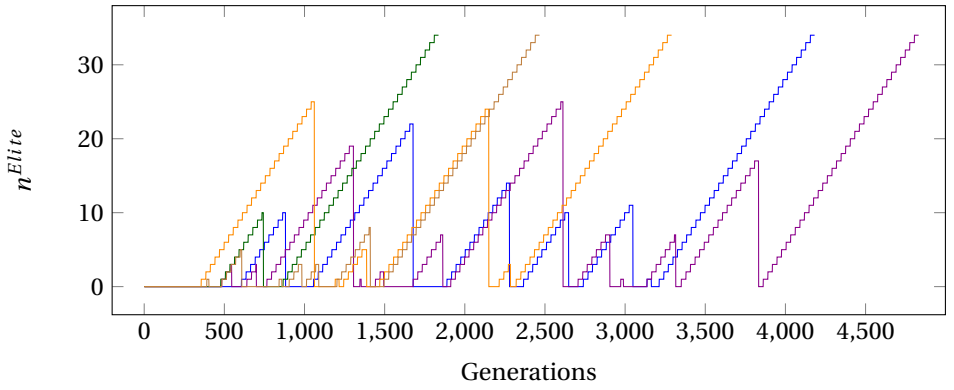

Figure 8.10: The value of n^{Elite} in the five runs on 51N373A as a function of generations, for the HGSDC with adaptiveness.

Table 8.13: Results from running the Basic HGSDC on Test set 5 when prohibiting U-turns. The asterisk marks that the run terminated as result of surpassing the maximum run time. This was a one-time occurrence when costs for U-turns were included.

	HGSDC with U-turns forbidden					
	Average		**Best**			
Instance	**Solution**	**Time(s)**	**Solution**	**Time(s)**	**RSD** $\times 100$	**Gap (%)**
51N373A	577	4,950	524	3,045	8.40	15.4
51N389A	582	3,728	554	4,789	5.77	14.2
51N406A	623	4,051	580	4,197	5.04	10.7
83N557A	847	3,196	776	2,996	9.61	20.1
83N562A	966	1,976	834	3,996	10.86	7.75
83N564A	950	2,036	825	3,227	12.37	27.5
123N829A	1,297	5,469	1,134	9,155	15.90	18.1
123N830A	1,260	5,270	1,095	10,000*	9.02	19.0
123N836A	1,394	5,160	1,207	5,411	11.90	17.8
Average	944	3,982	837	5,202	9.87	16.74

8.3.3 Prohibiting U-turns on large instances

Finally, the Basic HGSDC was run on Test set 5, while putting a cost on U-turns for the larger vehicles. The results are summarized in Table 8.13. *Gap (%)* is the percentage gap between the the best solution from the HGSDC with adaptiveness (z_A) and the best solution when adding a U-turns cost (z_U) as given by *Gap (%)* $= \frac{z_U - z_A}{z_A} \times 100$. In order to forbid U-turns, the cost per U-turn in the makespan solution was set to 1,000 time units. As the solution space when removing U-turns effectively is smaller than the solution space when they are allowed, it is expected that the makespan solution should be worse than those previously presented for the same instances. However, solutions obtained with no U-turns have an increased makespan of approximately 17% on average. Findings from previous studies on smaller instances tells us that the percentage obtained here is far too high. Although we do not know the optimal solution, we strongly suspect this to be an indicator that the heuristic is not particularly well-performing when disallowing U-turns in this way.

The relative spread of the heuristic solutions when penalizing U-turns, cannot be said to increase. Thus, it seems that the heuristic is fairly robust to such changes with respect to this specific measure. Computation time is on average increased when adding a cost for U-turns. The initial best solution generated usually contains loads of U-turns,

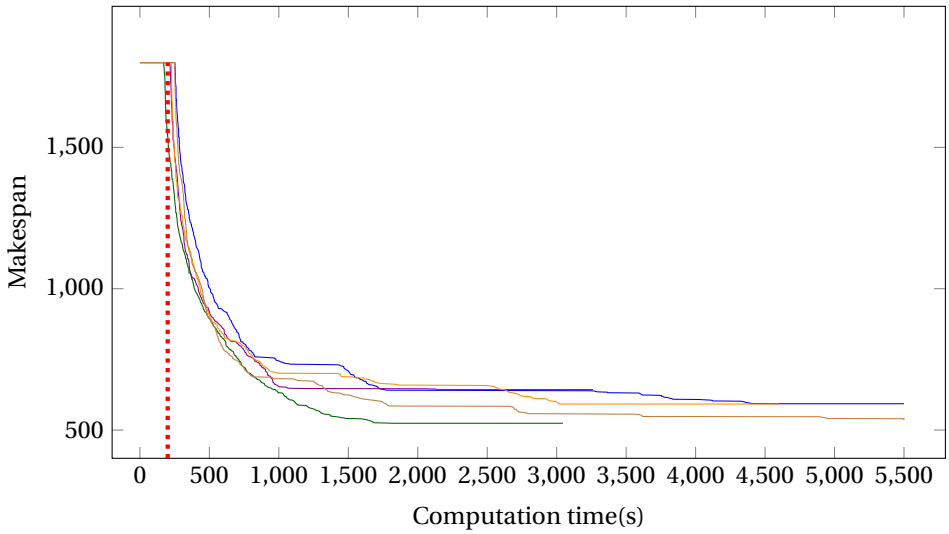

Figure 8.11: Makespan as a function on time for the Basic HGSDC when forbidding U-turns in instance 51N373A. The red dotted line at 200 seconds of computation time indicate where the burn-in period for all instances is over.

thus a burn-in period is needed in order to remove these solutions from the population, and start to improve "legal" solutions. Figure 8.11 shows the makespan of the best solution found over time for the heuristic with U-turn constraints, where the dotted line at 200 seconds of computations corresponds to the end of the burn-in period. Compared to the corresponding Figures 8.8 and 8.9 for the heuristic without the U-turn constraint the slopes of the curves after finding feasible solutions are also less negative, meaning that the U-turn constrained heuristic uses more time to improve the makespan solution. Neither this is unexpected, as the U-turn constraint removes a large part of the solution space, but not the search space. We therefore conclude that other approaches to removing U-turns should be considered.

Chapter 9

Concluding Remarks and Future Research

In this master's thesis, we have described the Multi-Fleet Arc Routing Problem with Synchronization Constraints (MFARPOS). The problem is found when planning routes for snow plowing vehicles in urban and residential areas. Special for the MFARPOS is the use of two homogeneous vehicle fleets - one fleet to service the lanes, and one to service sidewalks and pedestrian paths. As snow plowing vehicles push the snow to their right hand side, servicing some lanes will cause a snowbank to build up on the sidewalk. Therefore, the sidewalk should be serviced after the associated lane. This synchronization relation makes the fleets interdependent, and the MFARPOS a problem where computational effort grows rapidly with the size increase of the road network. The objective of the problem is to minimize the makespan of the routes generated, such that the road network is cleared in the least amount of time. To solve the problem, we have proposed and implemented a Branch-and-Price model, and a genetic algorithm, which have been tested on instances generated by an instance generator developed for the purpose of this thesis. The study has been carried out from a theoretical perspective.

Compared to a previously developed arc-flow model, implemented as a mixed integer program, the decomposed model did not yield very promising results. As there are few resource constraints to the labels in the subproblems, many label extensions are allowed. This increases the computation time in the subproblems, which again affects the Branch-and-Price algorithm's ability to solve large instances. When the path-flow

and arc-flow models were compared, the path-flow model's upper hand was its better handling of dense road networks, with a lot of synchronization. As the synchronization constraints to a larger extent complicate the MIP model than the Branch-and-Price algorithm, this was as expected. However, none of the models came close to solving instances with more than 20 nodes within a time limit of 36,000 seconds. This result speaks in favor of the development of heuristic approaches for the MFARPOS.

A genetic algorithm, the Hybrid Genetic Search with Diversity Control, based on the work of Vidal et al. (2012), was proposed and implemented. The heuristic, on average, yields an optimality gap of 5% for instances where the optimal solution is known. It is also able to generate routes on instances with up to 123 nodes and 800+ arcs, with 10 vehicles in both fleets; which is the size of large real-life instances. The framework that we have based our heuristic on, has unarguably proved to work well on different variants of the VRP, and we conclude that this thesis shows that such an approach is promising within ARPs as well. As infeasible solutions were disallowed in our implementation, we have proposed a new adaptivity measure, which changes the importance of makespan (in contrast to diversity contribution) in the population throughout the search. Although more testing needs to be carried out in order to validate these results, it seems that adapting the diversity may lower the spread, and thus yield more consistent performance. This indicates that for larger instances, a better balance between diversification and intensification in the solution space is needed. The heuristic is also capable of generating feasible solutions when forbidding U-turns, although with some increase in the computation time. For all variants of the heuristic proposed, solutions should be expected within 10,000 seconds.

Future research

There are many interesting areas for future development of the Branch-and-Price algorithm described in this thesis. The main problem uncovered with this implementation was the time used in the subproblems. Therefore, methods to reduce the computation time in the subproblems could make the algorithm more applicable for the MFARPOS. In order to do this, we propose using heuristic based methods to find good columns in the subproblems, and thereafter use the framework of the column generation as a matheuristic in order to find good integer solutions. Other dominance criteria should also be considered. With respect to branching, the B&B trees in some instances grew very big, although the root node solutions were close to optimal. We therefore recommend evaluating different branching strategies for this particular prob-

lem. As approaching ARPs with a Branch-and-Price solution methodology rarely has been attempted, and since our results show that such approach may indeed yield better bounds to instances where standard MIPs fail, we recommend further studying this framework on other ARPs. Finally, further improving the developed MIP model may also make this model able to solve larger instances to optimality. One such approach could be to explore possible valid inequalities for the problem.

As the computational results from the heuristic proposed in this thesis show that the framework proposed by Vidal et al. (2012) may also work well on Arc Routing Problems, we highly recommend considering similar genetic algorithm approaches to solve other variants of the ARP. Experimenting with the heuristic proposed in this thesis may also improve the search on the MFARPOS. Creating a specialized crossover operation dedicated for the MFARPOS would probably further improve the performance. With respect to adaptiveness, we recommend trying further evaluating the diversity strategy proposed, e.g. with an exponential or logarithmic increase instead of the proposed linear increase. Allowing for infeasible solutions, and thereby inherit the adaptive diversity control proposed by Vidal et al. (2012), should also be considered. In terms of special relations suited for real-life instances, more research on ways to include turn restrictions, would also greatly improve the relevance to the underlying real-life problem. Adaptation and implementation of other meta-heuristics, like the Adaptive Large Neighborhood Search, which has performed well on other problems concerned with snow plowing, may also work well on the MFARPOS.

References

Barnhart, C., Johnson, E. L., Nemhauser, G. L., Savelsbergh, M. W., and Vance, P. H. 1998. "Branch-and-price: Column generation for solving huge integer programs". In: *Operations research* 46.3, pp. 316–329.

Belenguer, J.-M. and Benavent, E. 1998. "The capacitated arc routing problem: Valid inequalities and facets". In: *Computational Optimization and Applications* 10.2, pp. 165–187.

Benavent, E., Campos, V., Corberán, Á., and Mota, E. 1990. "The capacitated arc routing problem. A heuristic algorithm". In: *Questiió: Quaderns dÉstadíistica, Sistemes, Informatica i Investigació Operativa* 14.1, pp. 107–122.

Blais, M. and Laporte, G. 2003. "Exact solution of the generalized routing problem through graph transformations". In: *Journal of the Operational Research Society* 54.8, pp. 906–910.

Bode, C. and Irnich, S. 2012. "Cut-first branch-and-price-second for the capacitated arc-routing problem". In: *Operations research* 60.5, pp. 1167–1182.

Borthen, T. and Loennechen, H. 2016. "The Multi-objective Supply Vessel Planning Problem-A Hybrid Genetic Search Approach". MA thesis. NTNU.

Christiansen, C. H., Lysgaard, J., and Wøhlk, S. 2009. "A branch-and-price algorithm for the capacitated arc routing problem with stochastic demands". In: *Operations Research Letters* 37.6, pp. 392–398.

Clossey, J., Laporte, G., and Soriano, P. 2001. "Solving arc routing problems with turn penalties". In: *Journal of the Operational Research Society* 52.4, pp. 433–439.

Desaulniers, G., Desrosiers, J., and Solomon, M. M. 2006. *Column generation*. Vol. 5. Springer Science & Business Media.

Drexl, M. 2012. "Synchronization in vehicle routing—a survey of VRPs with multiple synchronization constraints". In: *Transportation Science* 46.3, pp. 297–316.

Dror, M. and Langevin, A. 1997. “A generalized traveling salesman problem approach to the directed clustered rural postman problem”. In: *Transportation Science* 31.2, pp. 187–192.

Eiselt, H. A., Gendreau, M., and Laporte, G. 1995. “Arc routing problems, part II: The rural postman problem”. In: *Operations research* 43.3, pp. 399–414.

Euler, L. 1741. “Solutio problematis ad geometriam situs pertinentis”. In: *Commentarii academiae scientiarum Petropolitanae* 8, pp. 128–140.

Feillet, D., Dejax, P., and Gendreau, M. 2005. “The profitable arc tour problem: solution with a branch-and-price algorithm”. In: *Transportation Science* 39.4, pp. 539–552.

Floyd, R. W. 1962. “Algorithm 97: shortest path”. In: *Communications of the ACM* 5.6, p. 345.

Goldberg, D. E. and Lingle, R. 1985. “Alleles, loci, and the traveling salesman problem”. In: *Proceedings of an International Conference on Genetic Algorithms and Their Applications*. Vol. 154. Lawrence Erlbaum, Hillsdale, NJ, pp. 154–159.

Golden, B. L. and Wong, R. T. 1981. “Capacitated arc routing problems”. In: *Networks* 11.3, pp. 305–315.

Gundersen, A. H., Johansen, M., and Kjær, B. S. 2016. *Solving an Arc Routing Problem with Precedence Relations for Snow Plowing Operations*. NTNU.

Haarberg, K. and Aleksandersen, I. 2016. *Diskusjon rundt oppgave og dagens system*. Meeting. Tempevegen 22.

Holland, J. H. 1975. *Adaptation in Natural and Artificial Systems*. The University of Michigan Press.

Irnich, S. and Desaulniers, G. 2005. “Shortest path problems with resource constraints”. In: *Column generation*. Springer, pp. 33–65.

Kinable, J., Hoeve, W.-J. van, and Smith, S. F. 2016. “Optimization Models for a Real-World Snow Plow Routing Problem”. In: *International Conference on AI and OR Techniques in Constriant Programming for Combinatorial Optimization Problems*, pp. 229–245.

Lacomme, P., Prins, C., and Ramdane-Cherif, W. 2004. “Competitive memetic algorithms for arc routing problems”. In: *Annals of Operations Research* 131.1-4, pp. 159–185.

Lacomme, P., Prins, C., and Ramdane-Chérif, W. 2001. “A genetic algorithm for the capacitated arc routing problem and its extensions”. In: *Workshops on Applications of Evolutionary Computation*. Springer, pp. 473–483.

Lacomme, P., Prins, C., and Sevaux, M. 2006. “A genetic algorithm for a bi-objective capacitated arc routing problem”. In: *Computers & Operations Research* 33.12, pp. 3473–3493.

Laporte, G. 1997. “Modeling and solving several classes of arc routing problems as traveling salesman problems”. In: *Computers & Operations Research* 24.11, pp. 1057–1061.

Laporte, G., Musmanno, R., and Vocaturo, F. 2010. “An adaptive large neighbourhood search heuristic for the capacitated arc-routing problem with stochastic demands”. In: *Transportation Science* 44.1, pp. 125–135.

Lenstra, J. K. and Kan, A. H. G. R. 1976. “On general routing problems”. In: *Networks* 6.3, pp. 273–280.

Letchford, A. N. and Oukil, A. 2009. “Exploiting sparsity in pricing routines for the capacitated arc routing problem”. In: *Computers & Operations Research* 36.7, pp. 2320–2327.

Liu, T., Jiang, Z., and Geng, N. 2013. “A memetic algorithm with iterated local search for the capacitated arc routing problem”. In: *International Journal of Production Research* 51.10, pp. 3075–3084.

Pedersen, K. 2013. *Så mye snø kan du få i vinter*. YR. URL: http://www.yr.no/artikkel/sa-mye-sno-kan-det-komme-i-vinter-1.11312139 (visited on 09/28/2016).

Perrier, N., Langevin, A., and Amaya, C.-A. 2008. “Vehicle routing for urban snow plowing operations”. In: *Transportation Science* 42.1, pp. 44–56.

Perrier, N., Langevin, A., and Campbell, J. F. 2006. “A survey of models and algorithms for winter road maintenance. Part I: system design for spreading and plowing”. In: *Computers & Operations Research* 33.1, pp. 209–238.

Prins, C. 2004. “A simple and effective evolutionary algorithm for the vehicle routing problem”. In: *Computers & Operations Research* 31.12, pp. 1985–2002.

Riquelme-Rodríguez, J.-P., Langevin, A., and Gamache, M. 2014. “Adaptive large neighborhood search for the periodic capacitated arc routing problem with inventory constraints”. In: *Networks* 64.2, pp. 125–139.

Ropke, S. and Pisinger, D. 2006. “An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows”. In: *Transportation science* 40.4, pp. 455–472.

Salazar-Aguilar, M. A., Langevin, A., and Laporte, G. 2011. “An adaptive large neighborhood search heuristic for a snow plowing problem with synchronized routes”. In: pp. 406–411.

Salazar-Aguilar, M. A., Langevin, A., and Laporte, G. 2012. “Synchronized arc routing for snow plowing operations”. In: *Computers & Operations Research* 39.7, pp. 1432–1440.

Salazar-Aguilar, M. A., Langevin, A., and Laporte, G. 2013. “The synchronized arc and node routing problem: Application to road marking”. In: *Computers & Operations Research* 40.7, pp. 1708–1715.

Ting, C.-K., Su, C.-H., and Lee, C.-N. 2010. “Multi-parent extension of partially mapped crossover for combinatorial optimization problems”. In: *Expert Systems with Applications* 37.3, pp. 1879–1886.

Trondheim Kommune. 2016. *Brøyting*. URL: `https://www.trondheim.kommune.no/content/1117712643/Broyting` (visited on 10/01/2016).

Vidal, T., Crainic, T. G., Gendreau, M., Lahrichi, N., and Rei, W. 2012. “A hybrid genetic algorithm for multidepot and periodic vehicle routing problems”. In: *Operations Research* 60.3, pp. 611–624.

Appendix A

Details of the Branch-and-Price Results on Test Set 2

In Tables A.1, A.2, and A.3, a detailed description of the results when running the different versions of B&P algorithm on Test set 2 is shown. Table A.1 shows the results for the B&P algorithm with the cold started column generation, while Table A.2 summarizes the results when the column generation is warm started with the basic version of HGSDC heuristic, but none of the heuristic's solutions are added as columns. In Table A.3 the column generation is warm started, and the columns are added as well.

In order to calculate the gaps in the tables, the best dual bound known is used. *IP gap* measures the percentage gap from the best integer solution found within the respective algorithm, to the best known dual bound. That is, $IP\,gap = \frac{z^{IP}-z^*}{z^*}$, where z^{IP} is the best integer solution found by the respective algorithm, and z^* is the best known dual bound for the instance, rounded up to the nearest integer. Note that for all instances except 16N25A, the dual bound is optimal. *Lower gap* is the percentage gap from the dual bound obtained in the model to the best known dual bound; as calculated by $Lower\,gap = \frac{z^*-z^{LB}}{z^*}$, where z^{LB} is the best dual bound obtained by the model, rounded up to the nearest integer. *Heu gap* shows the percentage gap from the heuristic solution used as the warm start to the best known dual bound. That is, $Heu\,gap = \frac{z^{HS}-z^*}{z^*}$, where z^{HS} is the heuristic solution. *Root gap* refers to the percentage gap from the root node solution of the model to the best known dual bound. It is given by $Root\,gap = \frac{z^*-z^{RN}}{z^*}$, where z^{RN} is the root node solution, rounded up to the nearest integer.

Table A.1: Results of running Test set 2 in the B&P algorithm with cold start. *Total time*, *Time RMP*, *Time SPs*, *Time Heu* and *Time MIP* show the total time used, and the time used in the RMP, SPs, the heuristic, and the time spent solving the RMP as a MIP, respectively. *#Nodes in B&B* is the total number of nodes in the B&B tree, while *#Paths Sub* and *#Paths Heu* indicate the number of paths generated in the subproblem and heuristic, respectively. *n/a* indicates that the algorithm did not manage to prove optimality of the root node within the time limit. *) as the algorithm did not solve the root node to optimality within the time limit for test instance 16N52A, the results from this instance are not included in the average values.

Instance	Total time(s)	Time RMP(s)	Time SPs(s)	Time Heu(s)	Time MIP(s)	#Nodes in B&B	#Paths SPs	#Paths Heu	IP gap(%)	Lower gap(%)	Heu gap(%)	Root gap(%)
8N21A	2	0	1	-	0	21	392	-	0	0	-	0
8N28A	9	0	6	-	3	101	1,154	-	0	0	-	0
9N27A	162	12	91	-	3	1,391	5,575	-	0	0	-	1.25
10N38A	1,211	22	867	-	56	1,041	5,609	-	0	0	-	0
11N33A	2,292	29	1,491	-	762	4,155	5,945	-	0	0	-	10.42
12N37A	>36,000	7	35,918	-	76	335	3,567	-	8.11	1.80	-	6.31
13N46A	>36,000	28	35,301	-	76	541	6,330	-	6.67	2.22	-	2.22
14N46A	>36,000	2	35,996	-	3	13	2,234	-	29.03	5.38	-	5.38
15N44A	>36,000	21	35,866	-	115	107	5,748	-	8.04	18.75	-	18.75
16N52A	>36,000	92	35,909	-	0	1	10,309	-	n/a	n/a	-	n/a
Average*	>16,408	13	16,171	-	121	856	4,062	-	5.76	3.13	-	4.92

Table A.2: Results of running the B&P algorithm on Test set 2 with warm start without using any of the columns from the heuristic solution. *Total time*, *Time RMP*, *Time SPs*, *Time Heu* and *Time MIP* show the total time used, and the time used in the RMP, SPs, the heuristic, and the time spent solving the RMP as a MIP, respectively. *#Nodes in B&B* is the total number of nodes in the B&B tree, while *#Paths Sub* and *#Paths Heu* indicate the number of paths generated in the subproblem and heuristic, respectively. *n/a* indicates that the algorithm did not manage to prove optimality of the root node within the time limit. *) as the algorithm did not solve the root node to optimality within the time limit for test instance 16N52A, the results from this instance are not included in the average values.

Instance	Total time(s)	Time RMP(s)	Time SPs(s)	Time Heu(s)	Time MIP(s)	#Nodes in B&B	#Paths SPs	#Paths Heu	IP gap(%)	Lower gap(%)	Heu gap(%)	Root gap(%)
8N21A	44	0	0	43	0	1	204	-	0	0	0	0
8N28A	590	1	288	297	3	101	1,529	-	0	0	1.15	0
9N27A	269	0	0	269	0	1	480	-	0	0	0	1.25
10N38A	501	3	200	269	29	119	2,297	-	0	0	7.79	0
11N33A	819	5	373	334	107	441	2,637	-	0	0	8.33	6.25
12N37A	12,317	8	11,942	259	108	985	2,622	-	0	0	5.41	0
13N46A	>36,000	18	35,330	388	265	447	3,885	-	4.44	0	5.56	0
14N46A	>36,000	15	29,862	989	5,135	599	4,182	-	5.38	1.08	5.38	1.08
15N44A	>36,000	56	30,398	294	5,248	1,979	7,579	-	0	3.57	5.36	4.46
16N52A	>36,000	14	35,471	514	0	1	2,739	-	0.76	n/a	0.76	n/a
Average*	>13,616	12	12,044	349	1,211	519	2,824	-	1.09	0.52	4,33	1.45

Table A.3: Results of running the B&P algorithm on Test set 2 with warm start that includes the columns from the heuristic population. *Total time*, *Time RMP*, *Time SPs*, *Time Heu* and *Time MIP* show the total time used, and the time used in the RMP, SPs, the heuristic, and the time spent solving the RMP as a MIP, respectively. *#Nodes in B&B* is the total number of nodes in the B&B tree, while *#Paths Sub* and *#Paths Heu* indicate the number of paths generated in the subproblem and heuristic, respectively. The root node solution is rounded up to the nearest integer value. *n/a* indicates that the algorithm did not manage to prove optimality of the root node within the time limit.

Instance	Total time(s)	Time RMP(s)	Time SPs(s)	Time Heu(s)	Time MIP(s)	#Nodes B&B	#Paths SPs	#Paths Heu	Upper gap(%)	Lower gap(%)	Heu gap(%)	Root gap(%)
8N21A	43	0	0	43	0	1	77	175	0	0	0	0
8N28A	430	1	132	297	1	41	1,203	280	0	0	1.15	0
9N27A	149	0	1	269	0	1	705	280	0	0	0	0
10N38A	2,845	14	2,395	269	166	629	4,392	630	0	0	7.79	0
11N33A	761	4	384	334	38	321	2,314	455	0	0	8.33	6.25
12N37A	>36,000	60	22,770	259	12,893	3,749	6,590	455	5.41	0	5.41	0
13N46A	>36,000	5	35,596	388	12	47	3,229	630	5.56	0	5.56	0
14N46A	>36,000	21	32,218	989	2,772	561	4,573	630	4.30	1.08	5.38	1.08
15N44A	>36,000	185	21,515	294	13,972	6,623	9,906	630	0	3.57	5.36	4.46
16N52A	>36,000	39	35,446	514	1	11	7,710	630	0.76	2.27	0.76	2.27
Average	>18,423	33	15,446	366	2,985	1,198	4,070	480	1.60	0.69	3.97	1.53