Master's degree thesis

INF950 Masters Thesis

Time-dependent travel times - the case with ferries

Nils Ove Erstad

Stian Kroknes

Number of pages included the first page: 73

Molde, May 18, 2009

Molde University College

Publication agreement

Title: Time-dependent travel times - the case with ferries

Author(s): Nils Ove Erstad and Stian Kroknes

Subject code: INF950

ECTS credits: 30

Year: 2009

Supervisor: Arne Løkketangen

Agreement on electronic publication of master thesis

I/we hereby give Molde University College the right to, free of charge, make the thesis available for electronic publication: [✓ yes] no

Is there an agreement of confidentiality? yes [✓ no]
(A supplementary confidentiality agreement must be filled in)
- If yes: **Can the thesis be online published when the period of confidentiality is expired?** yes no

Date: 18.05.09

Abstract

In the Vehicle Routing Problem (VRP), not considering ferries when ferries are in fact present could cause the solution to be very inaccurate and very difficult to implement. As the travel time between two locations, when ferries are present, is dependent on the departure time, not considering ferries is in fact solving the wrong problem. This thesis focuses on the advantages of considering ferries in a VRP, as well as documenting the extra computational effort needed. We present an approach towards ferries using a set of travel times for each origin - destination pair containing one or more ferries. The travel times are calculated for departure times of chosen intervals using intermediate times to and from the ferry connections. Results show that considering ferries yield substantial improvements when implementing the route plans, in comparison to a standard VRP solver using static travel times.

Contents

1 Introduction

1.1 Motivation and background

During the time at Molde University College (HiM) we followed an introductory course given by professor Arne Løkketangen, *Heuristic Optimization Methods*. There we got to know the Vehicle Routing Problem (VRP) and when professor Løkketangen proposed a master thesis covering heuristic implementation of time-dependent travel times into an already existing VRP solver, we were really interested.

This master thesis covers the problem with time-dependent travel times in a VRP solver with focus on ferries. Not considering ferries in a VRP solver where ferries are in fact present, will cause the solver to be inaccurate as the travel time from an origin to the corresponding destination could be heavily dependent on the departure time from origin. Solutions provided by a standard VRP solver might look good, but in real life they may cause the vehicle to wait for a ferry when it could have visited another customer first or the solutions might even be infeasible. Adjusting the routes to connect better with the departure time of ferries could improve the solution.

Local topography is also a motivation for this thesis as the county we live in contains a lot of ferry connections. Only in our county, Møre og Romsdal, there are 38 ferry connections. For local companies, having a planning tool that considers ferries would be very beneficial.

Two local companies, Oskar Sylte and Nortura, have provided information used in this thesis. They have been kind enough to provide us with customer data used in their delivery plans as well as answered questions related to the problem formulated in this thesis. This thesis builds on the PhD work of Oppen (2008) and his work for Nortura. Oppen has also contributed by providing us with the source code of his VRP solver as well as valuable expertise and advice.

1.2 Outline of thesis

In Section 2 we will present the Vehicle Routing Problem as well as topics related to it. Section 3 looks at different ways to solve the VRP as it has been done previously, while Section 4 is about strategic choices made in this thesis. In Section 5 we present MIP models for the two problems as well as use of exact methods while 6 is all about the heuristic chosen in detail. Section 7 contains the computational experiments and results accumulated from this thesis. The conclusions are presented in Section 8.

2 Vehicle Routing Problem

The VRP is to optimally allocate transportation tasks to a fleet of vehicles, and finding an optimal routing for each vehicle thus minimizing costs. The VRP is of high industrial relevance and can arise in several industries. Examples of applications are the planning of local pickup and/or deliveries for transportation companies.

A classical capacitated VRP (CVRP) is defined over a graph $G = (N, A)$ where $N = \{0, ..., n\}$ is a vertex set (0 is the depot, other vertices are customers) and $A = \{(i, j) : i, j \in A, i \neq j\}$ is a set of arcs. The cost of traveling between customer i and j is defined as $c_{ij} \geq 0$ while d_i is the demand at customer i. Each vehicle also have a limited carrying capacity of Q. The goal is then to design optimal routes according to the objective, with all routes starting and ending at the depot. In addition all customers must be visited exactly once. Total demand of all customers in a route must be less than or equal to the capacity Q of the vehicle assigned to that route. If the total demand of the problem is larger than the aggregated capacity of the vehicles, the problem is infeasible (if not allowing multiple tours per vehicle).

2.1 Extensions

There are many extensions to the VRP, some of them need to be considered in this thesis. These extensions can also be combined to fit most real-world VRP problems.

One extension to the VRP is to include Time Windows (VRPTW). Time windows mean that the service at a certain location must start within the given time interval and the vehicle remains at the location for the duration of the service time. These time windows could also be applied to the depot, for example when a vehicle starts at 8:00am and must be back at the depot before it closes at 4:00pm. If we limit the time horizon this will in practice be a time window for the depot.

Another extension to the VRP is the introduction of backhauls (VRPB). This has two sets of customers, linehaul customers requiring a given amount of products to be delivered and backhaul customers who have inbound products to be picked up and brought back to the depot.

An extension similar to the VRPB is the VRP with pick-up and delivery (VRPPD). Here the customers have both demand for goods to be delivered and goods to be picked up. The vehicle must then pick up goods at one location and deliver it to a different location.

One extension included in this thesis is the time-dependent VRP (TDVRP). In TDVRP, the travel times between locations are dependent on the departure time of the vehicle. There are many reasons to why travel times are time-dependent, e.g. rush hour, weather, accidents and so on. The TDVRP deals with this having a set of travel times dependent on departure time, between all origin-destination pairs.

A more extended overview of these extensions to the VRP can be found in Toth and Vigo (2002).

2.2 Time-dependent travel times and costs

The purpose of this thesis is to look at the problem of time-dependent travel times with emphasis on inclusion of ferries. Time-dependent travel times means that the travel time from i to j depends on the departure time from i. Looking at ferries this means that although a ferry crossing has a fixed duration, the travel time is dependent on the vehicles arrival at the ferry docks. If the vehicle arrives just after a ferry leaves the dock, waiting time is until the next departure. Not looking at the stochastic aspects of ferry crossings (See Section 2.3), this problem has exact departure times and is therefore possible to implement in a standard VRP solver.

Another example of time-dependent travel time is where travel times, especially in urban areas, fluctuate due to rush hour congestion. At certain periods during the day, when roads are carrying more traffic, the travel time will be longer. Compared to ferries these times are less accurate, but using estimates based on statistics will make it possible to include this into a VRP solver.

There are also examples of time-dependent travel cost for using roads. This will not affect the travel time, but will give a larger cost for using roads at given time of day. As optimal routes in a VRP solver also can be evaluated from the cost of traveling, this could be of much importance. An example of this is the road pricing system of Bergen, Norway. Vehicles entering the city between 6:00am and 10:00pm on weekdays have to pay a charge. This way the cost for traveling on these roads are higher at certain times, making the cost time-dependent.

2.3 Stochastic time-dependent travel times

Most of the travel times in the real world are stochastic. Rush hour, weather, accidents and road work are all examples of why the travel time when traveling between two locations is stochastic. It is our belief that this is very difficult and time consuming to handle accurately in a VRP solver, although estimates can be used to model what is likely. Rush hour delays in cities are often similar from day to day and one can use real data to evaluate the congestion during the day. This can then be used as weights on the travel time on that link so that hours with heavy congestion have longer travel time than less congested hours.

Looking at ferries, there are also stochastic challenges, e.g. ferries not on schedule or ferries with limited capacity. If a large vehicle arrive at the docks just in time for a departure it might be the case that there is not enough space left on the ferry, especially on ferry connections carrying a great deal of traffic. If this is included in a VRP solver it could be assumed that the closer to a departure you arrive at the docks, the more likely it is that you could not embark. This could also be related to the time of day as with rush hour.

Weather is also a factor that can affect the travel times. If still looking at our county, winter conditions in combination with steep and/or curvy roads will give a longer travel time. Ferries are also affected by weather, especially wind. Heavy wind could cause the ferries to be off schedule or in worst case laid up. There are also examples of high and low tide causing problems for ferries loading and unloading because of the angle of the landing apron.

2.4 Ferries in VRP

As mentioned, not considering ferries in a VRP where ferries are in fact present, could cause the solutions to be very inaccurate. The ferry connections could be included with zero distance, the real distance of the crossing or even the sailing time. All would be inaccurate. It is possible that a ferry connection with frequent departures could be an approximation but still not accurate. For remote places with few departures a day this could in the worst case lead the solution to be infeasible. An example of this could be if a vehicle uses a ferry connection to a remote place but the service time at the destination is so large that the vehicle will not make the last return departure. The precision of the travel times is proportional to the density of ferries. In this thesis the density of ferries is defined as the share of arcs in the topology containing ferries e.g. 1 means that all arcs in the topology contain one or more ferry and 0.5 density means that half of the arcs in the topology contains ferries

Considering the time-dependent travel time aspect of ferries in a VRP, both when making the instances and when solving the problem will give more accurate solutions. Compared to a solution from a standard solver not considering ferries, taking the ferries into account may give a different solution. While the standard solution may choose a link containing a ferry that in real life would cause a lot of waiting time, the solver considering ferries might add one or two customers to the route before using the arc with the ferry. A time-dependent solver might also choose not to use the ferry at all. It could also be the case that a later departure from the depot will be beneficial. This way, some of the solutions, maybe even the optimal solution for a standard VRP could be rendered infeasible or at least far from the real optimum.

When asking the two companies providing customer data for this thesis, Nortura and Oskar Sylte, on how they solve the problem with ferries today, the answer from both were that this is something that is done by the drivers themselves. They both state that they have not experienced any problems due to bad planning of ferries. The exception for this are stochastic problems like weather, for example when all ferries are laid up because of storms. This claim was in a way surprising as the geographical area covered contain loads of ferry connections. One statement gave away clues that these routes might be carefully planned as they said that their drivers "always give themselves more time on routes with ferries".

This could mean that the routes contain less customers than it could have, and/or that the driver use more time on the route compared to an optimal solution considering ferries. This would again lead to larger driver and traveling costs for the companies.

3 Solution methods

This section covers some of the earlier work done on solving the VRP, both with and without time-dependent travel times. Many methods are proposed to solve a standard VRP while there is less extensive research done on the case of time-dependent travel times. There is not much work published on the VRP with ferries.

3.1 Exact methods

Laporte and Nobert (1997) gives a overview over exact methods to solve a VRP problem. They cover methods like *Assignment lower bound*, *k-degree center tree*, *Dynamic programming*, *Set partitioning and column generation*, *Two-index vehicle flow formulation* and *Three-index vehicle flow formulation*. Common for all these exact methods is that they are only capable of solving smaller problem instances of size up to 50 nodes.

3.2 Heuristic algorithms

Heuristics are used to solve larger instances. Heuristics methods can not guarantee solving a problem to optimality, nor can it prove optimality when an optimal solution is actually found. Heuristic methods can be able to produce very good results close to the optimum very quickly thus they are often chosen before exact methods to solve large problem instances. In many cases it is more important, and even beneficial, to get good solutions quickly rather than a near-optimal or optimal solution after a long time. Heuristics are commonly divided into two parts, classical heuristics and metaheuristics (outlined in Section 3.2.2).

3.2.1 Classical Heuristics

The most popular classical heuristics are naturally divided into two groups, *constructive heuristics* and *improvement heuristics*.

Clarke and Wright (1964) introduced a savings algorithm, a constructive algorithm for the Traveling Salesman Problem (TSP) with the basic idea of creating small routes to all nodes in the graph and then merge these one by one by choosing the feasible merge that will provide the largest savings. This is then repeated until they could not save more, having a feasible solution to the TSP. The TSP is another combinatorial optimization problem where the task is to find a shortest possible tour that visits all customers exactly once. A VRP with number of vehicles equal to one is a TSP. The savings algorithm can also be adapted to the CVRP where customers are inserted into several routes according to the largest saving and the capacity constraint.

Gillett and Miller (1974) proposed another constructive heuristic using a sweep algorithm. Starting with a half-line rooted at the depot, this heuristic gradually constructs feasible routes by rotating a second half-line. Customers are gradually incorporated into the current route in increasing order of the angle they make with the initial half-line. The route closes when the inclusion of a further customer becomes infeasible due to capacity constraints.

Laporte (2007) describes two types of improvement algorithms that can be applied to VRP solutions, *Intra-route* and *Inter-route* heuristics. Intra-route heuristics post-optimize each route by using TSP improvement heuristic e.g. 2-opt or 3-opt as presented by Kernighan and Lin (1973). Inter-route heuristics consist of moving vertices to different routes. Laporte also states that the most common moves are simple transfers from one route to another and transfers involving several routes and vertex exchanges between two ore more routes. The performance of classical improvement heuristics is good but not excellent and are best used as building blocks within metaheuristics or to get initial solutions as starting points for metaheuristics.

3.2.2 Metaheuristics

Metaheuristics are heuristics that explore the solution space beyond the first local optimum encountered. All metaheuristics adapt procedures from classical heuristics and are broadly classified into three categories in Laporte (2007); *local search*, *population based search* and *learning mechanisms*.

A local search heuristic start from a initial solution (could be any solution, even infeasible) and moves to a different solution in the neighborhood. The neighborhood of a solution is defined as all the solutions one can reach by one transformation, e.g. a move of a customer to a different route. *Tabu Search* by Glover and Laguna (1997) is a very popular local search based metaheuristic and has proven to be very successful. The idea of tabu search is to make recent transformations taboo for a certain number of iterations to avoid cycling.

Using tabu search on the VRP is well-studied and has much published work. Many implementations of the tabu search metaheuristic has been successful and has provided good results. Some that can be mentioned are the approach by Taillard (1993) using two partition methods (problems with polar regions or of arborescent form), the *Taburoute* by Gendreau et al. (1994) and *Unified Tabu Search Algorithm* by Cordeau et al. (2001). An overview of these algorithms and their performance as well as more can be found in Gendreau and Laporte (2005).

One population search based algorithm is *Genetic Algorithms (GA)* by Holland (1975). GA is inspired by evolutionary biology and evolve a population of solutions represented by chromosomes through a crossover and mutation process. A common structure of GA is; First, the crossover takes two parents and combine them to generate one or two offspring chromosomes. A mutation process is then applied to the offspring and the offspring replace the worst element in the population. Another very good memetic (GA in combination with a local search heuristic) algorithm is that of Nagata (2007) who initially relaxes the capacity constraint and handles it through a penalty function when exploring neighborhoods. Another approach that has proven to yield very good results is the approach by Mester and Bräysy (2007) where an active-guided evolution strategy is used on the CVRP. Mester and Bräysy (2007) combines the strengths of the well-known guided local search and evolution strategies metaheuristics into an iterative two-stage procedure.

Ant Colony Optimization (ACO) is one metaheuristic classified as reinforcement learning mechanism. This is also inspired by nature and attempt to mimic the behavior of ants. Ants detects paths containing pheromones and strengthen it with it's own pheromone. As ants choose the shortest paths, the pheromone on these accumulates faster. In ACO the system memory is the pheromone and represents edges often appearing in good solutions making good edges more likely to appear in good solutions.

3.3 Time-dependent travel times

The case with time-dependent travel times is when the travel time between two customers can vary dependent on the departure time. The change in travel time could be due to road congestion, ferry crossings and so on.

Malandraki and Daskin (1992) propose an ILP formulation of the time-dependent VRP and TSP. Each link has a cost that is a step function depending on the time of departure. The paper also gives two heuristics for the time-dependent VRP based on tour construction and a heuristic cutting-plane approach. The problem with step functions is that the can be cases where the non-passing property (FIFO) is violated. This property states that if vehicle A depart at the end of a step where the cost is high and another vehicle B depart in the beginning of the next step where cost is low, vehicle B might arrive at the destination first.
Ichoua et al. (2003) formulated a model based on time-dependent travel speeds. They adjust the speed of the vehicle by dividing the time horizon (a day) into time periods, thus changing the speed when the boundary between two consecutive time periods is reached.
Donati et al. (2003) combines robust shortest path (RSP) algorithm and ant colony optimization. The RSP algorithm uses an interval representing the possible travel times for each arc. The algorithm then use the time-dependent interval data to get more reliable travel time thus it is more robust than the normal shortest path algorithms. They apply the method to a VRP based on the Padua road network and dynamic vehicle speeds that were collected hourly from the traffic control system Cartesio. The robust shortest path for each pair of customer locations is precomputed and stored in memory before the optimization starts.
Haghani and Jung (2005) also present an ILP formulation of a pick-up or delivery VRP with soft time windows in which they consider multiple vehicles with different capacities, real-time service requests, and real-time variation in travel times between demand nodes. They use a continuous travel time function instead of a step function, a way that satisfies the non-passing property.
Kerbache and van Woensel (2004) model a VRP with time windows and stochastic travel times. Their approach handle the potential traffic congestion using queuing theory to capture the stochastic behavior of the travel times. In addition they use a case study to show that time-independent solutions often are unrealistic within congested traffic environments.
Donati et al. (2008) present a Multi Ant Colony System for the time dependent VRP. They

focus on variable traffic conditions on real road networks, like in urban environments. They also formulate a time dependent local search procedure and they provide computational results of the use of RSP algorithm.

4 Solution strategy and implementation techniques

As well as getting a good problem description, the choice of strategy and techniques are also important. Presented in this Section are the strategic choices made regarding strategy towards inclusion of ferries as well choices made for implementation.

4.1 Solution strategy

In a normal network topology the link between two vertices would be represented as one edge. In real life travel from i to j will normally contain a path of several consecutive arcs. The distances of the consecutive arcs would make the total distance from i to j. As some of these arcs could be ferries these have to be treated differently. This would also mean that the graph is not symmetric as the travel time from j to i could differ from the travel time from i to j. The main idea for this thesis is to identify connection (i, j) that contains one or more ferries and use intermediate travel times to and from the ferry docks to find the next departure time for the ferry.

Looking at a real world problem, intermediate times would then mean the time used on each Section of the route that does not contain a ferry. Going from i to j with one ferry connection on the route, this will give us two intermediate times. From i to the first ferry dock and from the dock to j. This is illustrated in Figure 1. If there are several ferry connections the first step can be repeated by storing intermediate time from the dock where the vehicle disembark the first ferry and to the next ferry dock.

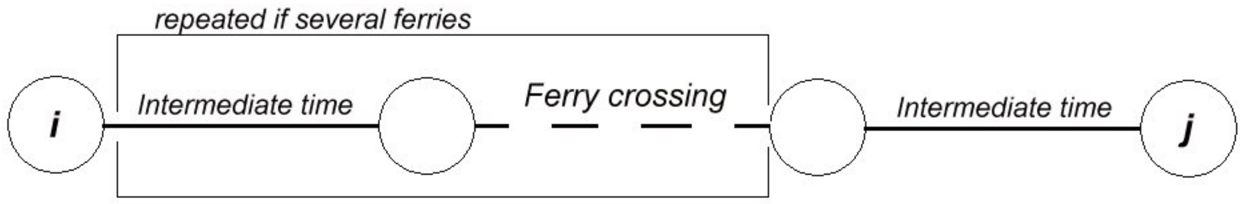


Figure 1: Illustration of intermediate times before and after ferries

The idea is to get an Origin-Destination (O-D) matrix with all travel times from origin i to destination j, where all routes containing ferries are specially marked and the rest have the normal static travel times. First the day is partitioned into p time intervals $T_1, T_2, ..., T_p$. Travel times for each route containing ferries are then calculated for all time intervals. By using the intermediate times accumulated we then find the travel time to the first ferry encountered. The ferry's timetable is then processed to find the next possible departure and time is set equal to that departure. Finally the ferry's crossing time is added. This is then done for all ferries in the route (if more), and the time from the last ferry to the destination is added.

The resolution of T, the number of intervals (p), is an important issue to address when using this strategy. Storing the time-dependent travel times for all arcs (i, j) that contain ferries leads to a growing size for the data file as p, the number of intervals, gets bigger. Similarly, the file size is reduced as the p is reduced. The same principle is valid for the time needed to generate the instances with time-dependent travel times and reading them into the TDVRP solver. The precision of the data is also dependent on p as a small p gives more precision than a large p. A small p could for example mean that the time-dependent travel times are calculated for each minute throughout the time horizon, while a large p could be for every 15 minute.

Results from tests regarding intervals are presented in Section 7.

4.1.1 Advantages and drawbacks of the strategy

The strategy using intermediate times to and from ferries is an approach which includes ferries in a VRP solver without adding much to the complexity of the solver. Where standard VRP solvers have one lookup for travel times from i to j, the approach in this thesis needs two; first to retrieve the time-dependent travel times for (i, j) and then the travel time for the right interval. This is also the case if there are several ferry connections on the arc (i, j). Maintaining the level of complexity in the solver indicate that ferries are implemented efficiently. Compared to the benefits of getting more accurate solutions when considering ferries (see Section 2.4), the computational effort needed for the extra lookup would be insignificant.

The strategy proposed also have some issues that could be considered as drawbacks. The most significant is the possibility of alternative routes outlined in Section 4.1.2. The strategy proposed focus on the inclusion of ferries in the shortest paths found for the VRP, although inclusion of ferries could lead to other routes being shortest paths for certain time intervals. Using the shortest paths could also result in several ferries on the same arc (i, j). This is not considered a problem as it is assumed that the ferries are there to provide the shortest path to the destination and that the interval in which the ferries depart is the optimal schedule to provide this shortest path.

Another less important drawback with this strategy is the aspect of waiting time. Considering ferries on an arc (i, j) will in most cases lead to some waiting time. This waiting time will be represented as waiting time on the arc (i, j) for the given interval p. If there is only one ferry on the arc, the waiting time for (i, j) will be the waiting time before that ferry. The drawback is when there are several ferries on the arc, the waiting time will be the accumulated waiting time before all ferries.

4.1.2 Other strategies towards ferries in VRP

There are several ways to approach the problem with inclusion of ferries in a VRP. One that has been discussed is to use the same approach as used in this thesis but to handle the ferries while deciding shortest paths between vertices, not after the decision is made. The purpose is then to find the shortest path with relation to time, for all time intervals, where there are ferries. In that way the algorithm may find an alternative route that will give lower travel time than waiting for a ferry and using the ferry connection. It is our belief that this would be very time consuming. This would require that the algorithm not only search for shortest paths between all vertices but also for all time intervals, e.g. 1440 intervals per origin-destination pair if 1 minute intervals are used.

As mentioned, the approach used in this thesis has one major drawback. It has no possibilities for choosing an alternative route to avoid waiting for a ferry. In some cases this could cause some problems, for instance having a route over several days when ferries has less frequent departures at night. In this thesis we assume that taking such an alternative route would not be beneficial if there are no customers along the alternative route. We also assume that this problem can be worked around automatically by the solver as it can add customers prior to a ferry departure or even choose another sequence of customers so that it will become an alternative route not needing to use the ferry connection.

Another approach is to let the solver find shortest paths and handle ferries simultaneously. Providing the solver with not only the customer data, but the entire topography with several road links between customers, could give the solver a chance to find optimal routes considering ferries. With this approach, handling of stochastic behavior through "live update" of for example ferries off schedule or cancellations is possible. Using this approach will mean that we will get a very complex solver that would need to handle so much more than a standard solver normally do. We believe that this approach would need a lot of programming and would be very heavy computationally at run-time, although the travel time matrix can be built gradually during the search.

4.2 Metaheuristic

Tabu Search is the metaheuristic chosen for this thesis. There are several reasons for this choice. Prior knowledge is the biggest contributing factor as Tabu Search was the heuristic used in the course project where we first were introduced to VRP and heuristics. This will take some of the focus away from building the solver and over to the problem at hand. As mentioned we also have access to Johan Oppen and his PhD work Oppen (2008) and we found it appropriate to use the same heuristic as some of the results of our work might also be implemented in the solver used by Nortura, if found satisfactory.

The history of the Tabu Search is also a major factor in our choice. It has been widely used on VRP and has proven to yield good results. Bräysy and Gendreau (2001) has tested several of the implementations of Tabu Search and give an overview of the basic features as well as a presentation and analyzes of the experimental results from the tested implementations.

4.3 Implementation language

As programming language for implementing the solver C++ was chosen. This language is popular in the field of operational research, especially when the problem is demanding fast execution. The solver by Oppen (2008) is also implemented in C++ thus using C++

for this thesis would ease the future integration. As we had no prior knowledge of the language we were also curious to try it.

4.4 Data structures

To minimize the extra effort needed to get the travel time for arcs with ferries, the best possible data structure to use in the time-dependent VRP solver needs to be identified. The implementation of a standard VRP solver uses a two-dimensional vector of doubles or ints to hold the travel times. This vector operates in constant time for lookup given an O-D pair. The goal is then to find a data structure that is as close to this time as possible for the extended lookup needed when ferries are present. More detailed information on the mentioned data structures can be found in the *C++ Library Reference* (http://www.cplusplus.com/reference/).

In the time-dependent solver, retrieving travel times from O-D pairs that does not contain ferries will approximately be in constant time. This is not the case when the O-D pair contains one or more ferries as it needs to be checked for the right interval p and its associated travel time. As this is the main difference from the standard solver we have compared the use of *C++ STL Vector* and *Map* (*C++ Library Reference*) for storing the travel intervals and using *double* or *int* for the time variables.

For the VRPTD solver, *vector* has been chosen as the data structure. A primary two-dimensional vector contains objects for each O-D pair. This object holds another vector containing the travel intervals with its associated travel time and distance.

Computational results from the data structure testing can be found in Appendix B.

5 Mathematical model and exact methods

Making mathematical models for the two problems, CVRP and TDVRP, will be beneficial in several ways. First of all, building the solvers would be easier when having exact models to build from. Solving the problems exactly can only be done on smaller instances, or lower bounds for larger instances can be provided by running for a certain period of time. For this thesis the purpose of using exact methods is to get optimal solutions on small instances to validate the standard VRP solver as well as getting a good definition of the problem before creating the solvers.

5.1 MIP model for CVRP

The model presented here is based on standard models of the VRP problem as by Toth and Vigo (2002). In comparison to standard models, the one presented here has the extension of sub-tour elimination by using the MTZ (Miller-Tucker-Zemlin) constraint by Miller et al. (1960). Sub-tours are disjoint tours, in other words tours that are not connected to the depot. The instances only need to be solved once, in comparison to other sub-tour eliminating methods that need constraints to be added for each sub-tour found and then solved again.

$$min \sum_{(i,j)\in A} c_{ij} X_{ij} \tag{1}$$

$$s.t.$$

$$\sum_{(s,j)\in A} X_{sj} = |K| \tag{2}$$

$$\sum_{(i,j)\in A} X_{ij} = 1, \forall i \in V \setminus \{s\} \tag{3}$$

$$\sum_{(i,j)\in A} X_{ij} = \sum_{(j,i)\in A} X_{ji}, \forall i \in V \tag{4}$$

$$U_j \leq U_i - d_i + Q(1 - X_{ij}), \forall i, j \in V \setminus \{s\}, i \neq j \tag{5}$$

$$0 \leq U_i \leq Q, \forall i \in V \setminus \{s\} \tag{6}$$

$$X_{ij} \in \{0, 1\}, \forall (i, j) \in A \tag{7}$$

Here, V is a set of all nodes including the depot, A is a set of arcs between nodes i and j and K is the set of vehicles. The parameter c_{ij} is the cost of using the arc between i and j while X_{ij} is a binary decision variable taking the value 1 if the arc between i and j is used and 0 otherwise. Parameter s represent the depot node while Q is the vehicle capacity. U_i is a monotonous decreasing load on vehicle after visiting customer i and d_i represent the customer demands.

The objective function (1) expresses the minimization of the total travel cost. Constraint (2) expresses that all vehicles must start at the depot while (3) expresses that all customers must be visited. (4) makes sure that the in-degree is equal to the out-degree for all nodes. (5) is a sub-cycle eliminating constraint making sure that we have no routes disconnected from the depot. As the load at j is restricted to be less than the load at j in addition to the demand at j for all customers, no routes can be disconnected from the depot as those routes would violate the constraint. (6) are bounds for the load on each vehicle while (7) impose binary constraints on the X_{ij}variables.

5.2 MIP model for the TDVRP

A mathematical model has also been developed for the TDVRP, in order to get a good problem definition.

$$min \sum_{k \in K} \sum_{(i,j) \in A} (c_{ij}(T_{ik}))X_{ijk} \tag{8}$$

$s.t.$

$$\sum_{k \in K} \sum_{(i,j) \in A} X_{ijk} = 1, \forall i \in V \setminus \{s, \bar{s}\} \tag{9}$$

$$\sum_{(s,j) \in A} X_{sjk} = 1, \forall k \in K \tag{10}$$

$$\sum_{(i,\bar{s}) \in A} X_{i\bar{s}k} = 1, \forall k \in K \tag{11}$$

$$\sum_{(i,j) \in A} X_{ijk} = \sum_{(j,i) \in A} X_{jik}, \forall i \in V \setminus \{s, \bar{s}\}, k \in K \tag{12}$$

$$\sum_{i \in V \setminus \{s,\bar{s}\}} \sum_{(i,j) \in A} d_i X_{ijk} \leq Q, \forall k \in K \tag{13}$$

$$T_{sk} = E, \forall k \in K \tag{14}$$

$$T_{jk} - c_{ij}(T_{ik}) - T_{ik} \leq (1 - X_{ijk})M_{ij}, \forall (i,j) \in A, k \in K \tag{15}$$

$$E \leq T_{ik} \leq L, \forall i \in \{s, \bar{s}\}, k \in K \tag{16}$$

$$X_{ijk} \in \{0, 1\}, \forall (i,j) \in A, k \in K \tag{17}$$

Here, V is a set of nodes including the depot, A is a set of arcs between i and j while K is a set of vehicles. $c_{ij}(T_{ik})$ denotes the time-dependent travel time when vehicle k departs from node i while T_{ik} is the variable holding the vehicles departure time from i. X_{ijk} is a binary variable taking the value 1 if vehicle k travels from node i to node j and 0 otherwise. To control the time variable we need to make a copy of the depot node as this will have two recorded times for each vehicle, departure and arrival. If only one depot node is used in a tour, constraint (15) will always be violated as T_{sk} can not be both smaller than the T_{jk} for the first customer visited and larger than T_{ik} for the last customer visited by vehicle k. The two depot nodes are represented as s and $\bar{s}$. The demand at customer i is denoted as d_i while the vehicle capacity is given as Q. M_{ij} is a big number used to control the continuity of the time variable.

The objective function (8) expresses the minimization of the total travel costs. (9) makes sure that all customers are visited by one and only one vehicle. Constraints (10) and (11) expresses that each vehicle must start and end at the depot (end at copy of depot node) while (12) is a continuity constraint stating that a vehicle visiting a customer must also leave it. (13) expresses the capacity for the vehicles. (14) makes sure that the time variable for each vehicle starts at the start of the time horizon when leaving the depot. Constraint (15) controls the time variable using big M notation. In short it states that the departure from j must be equal to the departure from i plus the travel time from i to j if the arc (i, j) is used by vehicle k. (16) limits the time variable to be within the time horizon where E is the earliest start time possible and L the latest arrival back at depot, while (17) impose binary constraints on the X_{ij} variables.

5.3 Comparison of models

The main difference in the model in Section 5.2 from the one shown in Section 5.1 is the time variable. As the travel times are time-dependent we need to ensure that the travel time calculated is the one corresponding to the vehicles departure time. This time variable also require some constraints to make sure that the time is continuous, meaning that the arrival at j is equal to the departure from i and the cost of traveling from i to j. The sub-eliminating constraint from the CVRP model is also removed, but this is now controlled by the constraints (10) and (11) forcing all tours to start and end at the depot.

5.4 Solving with CPLEX

The model for the CVRP (see Section 5.1) have been coded for AMPL and solved using CPLEX as solver. The purpose is to obtain optimal solutions for small problems that can be used to validate the CVRP solver. The model for the TDVRP has not been coded for AMPL as the time dependency of travel times c_{ij} causes the problem to be non-linear. The model for the TDVRP has only been used as problem definition.

6 Tabu search heuristic

As mentioned in Section 3, tabu search (TS) is a metaheuristic that explores the solution space by using local search. TS explores the solution space by moving from one solution s to the best solution in its neighborhood $N(s)$ at each iteration. To avoid cycling there must be some anti-cycling rules. This is done by declaring attributes of recent moves performed tabu for the length of the *tabu tenure* θ, where θ is the number of iterations the customer is not allowed to be moved back to the same route. To further explore the solution space, a diversification strategy is used to guide the search into other regions of the solution space. This can be based on frequency based counters on attributes of solutions, e.g how many times a customer has been added to a tour or how many times a customer has been moved in total.

6.1 The Tabu Search algorithm implemented

The implementation of TS in this thesis is based on Cordeau et al. (2001) and ideas from Oppen and Lokketangen (2008). Let $c(s)$ denote the total travel cost for the routes in a solution $s \in S$ and $q(s)$ denote the violation of the capacity constraint. The violation of the capacity constraint is computed for each route with respect to the vehicle capacity Q. A solution is evaluated by the cost function (18), where α is a dynamically adjusted parameter.

$$f(s) = c(s) + \alpha q(s) \tag{18}$$

As the parameter α is dynamically adjusted with respect to the load constraint, this allows exploration of infeasible solutions. This is performed after each move. If the move leads to a feasible solution, the value of α is decreased to make it cheaper to visit an infeasible solution. Whenever the move lead to an infeasible solution the value of α is increased to make it more costly to visit infeasible solutions, thus guiding the search back to the feasible region of the search space.

An attribute set $A(s) = \{(i,k) : \text{ customer } i \text{ served by route number } k\}$ which is associated with each solution $s \in S$ is used for the diversification mechanism. The neighborhood $N(s)$ of a solution s is defined by applying an operator that removes an attribute (i,k) from $A(s)$ and replace it with a different attribute (i,k'), where $k \neq k'$. The size of the neighborhood can then be expressed as $|N| = n(m-1)$.

As proposed by Oppen and Løkketangen (2006), a 2-opt post-optimization procedure is performed on solutions that are good. A solution is good if it is feasible and it has a total cost less than η times the cost of the current best solution and the algorithm has performed at least 100 iterations. Oppen and Løkketangen (2006) recommend $\eta = 1.1$. For this thesis, the value of η is presented and tested in Section 6.2.

6.1.1 Moves

A move is performed as a simple transfer of a customer i from one route k to a different route k'. When customer i is removed from route k, the route k is reconnected by linking the preceding customer with the successor of the moved customer. Customer i is inserted in route k' between two consecutive customers. The position is determined by the move that yields the smallest value of the move evaluation function (18).

After transferring customer i from route k to route k', moving the customer back to the same route k will not be allowed (is tabu) for θ iterations by assigning a tabu status for the attribute (i,k). The only exception is when the aspiration criterion is met, that is the move yield a solution with a lower cost than the current best solution.

6.1.2 Diversification

Two mechanisms for diversification has been implemented. The reason why two mechanisms were chosen was the ease of implementation in combination with the possibility of finding the best strategy for this thesis.

Oppen and Løkketangen (2006) diversify the search by giving a penalty $p(s') = \lambda \sum_{(i,k)\in A(s')} \rho_{ik}$ to the $f(s')$ for any solution $s' \in N(s)$ such that $f(s') \geq f(s)$. ρ_{ik} is a number indicating how many times the attribute (i, k) has been part of a good solution (see Section 6.1). The intensity of the diversification is controlled by the parameter λ.

The second mechanism implemented is the method proposed by Cordeau et al. (2001). They penalize the value of $f(s')$ only if $f(s') \geq f(s)$ by a factor that is proportional to the number of times the attributes (i, k), in any solution $s' \in N(s)$, have been added to a solution. This is then multiplied with a scaling factor $\sqrt{nm}$ where n is the number of customers and m is the number of vehicles. Here, ρ_{ik} is the number of times attribute (i, k) has been added to the solution during the search process. The penalty $p(s') = \lambda c(s')\sqrt{nm} \sum_{(i,k)\in A(s')} \rho_{ik}$ is added to the value of $f(s')$. λ is a parameter used to control the intensity of the diversification.

Both these mechanisms drive the search process to less explored regions of the search space whenever the search has encountered a local optimum. Note that the penalty is not added to $f(s')$ if $f(s') < f(s)$.

6.1.3 Solver specific details

A few choices have been made regarding the test cases generated in this thesis. These choices are needed to reproduce the results presented in Section 7.

- Start of time horizon is set to 9:00 AM and there is no constraint on route duration.
- Service times at customers are included in the travel times.

There are also details about the two solvers worth noting:

- Both solvers are purely deterministic, no randomization is used.
- The TDVRP solver has an option to run post-optimization on routes where the departure time can be adjusted by ± 1 hour to check if lower total travel time can be obtained. The value of 1 hour is chosen based on our judgement of what is reasonable in the real world. In reality, this value would have to be chosen by the company doing the vehicle routing.

- After a move is performed, the total time for the whole tour is recalculated (for simplicity this is done for both the CVRP and TDVRP solver).
- All time-dependent travel times are calculated and stored before running. When using span, the spans are stored with it's travel time. During lookup for travel time for an interval, the right span have to be identified and travel time calculated.
- When using span, an index pointing to the previous span used is used for each origin-destination pair. The index is initially set to the span containing the interval equal to the start of the time horizon.

6.2 Preliminary testing

Many parameters are used to guide the tabu search. Even though this thesis focus on presentation and solution mechanisms for the inclusion of ferries in a VRP-Solver, finding the parameters that would yield the best results would be of value.

6.2.1 Test cases

To do preliminary testing and to validate the implementation of the standard solver it has been used a subset of instances for the CVRP found at the web site *Branch Cut and Price Resource Web* (http://branchandcut.org/VRP/data). For most of the instances the optimal value is known, but for the rest it is provided a best known solution value for comparison. The subset used for parameter testing contains the instances *A-n32-k5.vrp*, *B-n50-k7.vrp*, *E-n101-k14.vrp*, *F-n135-k7.vrp* and *P-n70-k10.vrp*. The selected set has instances with customers varying from 31 to 134 and number of vehicles needed varying from 5 to 14. Using a varied test set give better indication of solver performance relative to the tested parameters. Note that the euclidean distance calculated for the instances in the subset are rounded off to the nearest integer.

Preliminary testing for the TDVRP solver has been done on the test cases made from real world data (see Section 7.2.3).

6.3 Parameter values

Parameter sets has been generated as all combinations of selected parameter values for testing. The parameter values selected for testing is presented in Tables 1 and 2. Table

1 show the tested parameter values using the geometrical infeasibility strategy (see 6.3.3) from Cordeau et al. (2001) while Table 2 show the parameter values tested using the arithmetical strategy proposed by Oppen and Løkketangen (2006). In the tables, the diversification mechanism by Oppen and Løkketangen (2006) is represented as DivMode equal to 1, DivMode 2 represent the diversification mechanism by Cordeau et al. (2001).

Table 1: Parameters tested for geometrical infeasibility strategy.

DivMode	$\delta+$	$\delta-$	λ	α	η
1	1.02	1.02	0.015	1	1.01
2	1.5	1.5	5	5	1.05
	5	5	50	10	1.1
	10	10	10	100	10
	100	100			

Table 2: Parameters tested for arithmetical infeasibility strategy.

DivMode	$\delta+$	$\delta-$	λ	α	η
1	0.01	0.01	0.015	1	1.01
2	0.05	0.05	5	5	1.05
	0.5	0.5	50	10	1.1
	5	5	10	100	10
	10	10			

For each infeasibility strategy the total number of different parameter sets are 3200. As the number of test cases for preliminary testing are 5 (see Section 6.2.1), the total number of test runs required for the standard solver are 32 000. For the TDVRP solver, the total number of test runs required are 38 400.

6.3.1 Run time

All of the selected test instances have been run for 5 minutes, regardless of problem size. The only exception is that when known optimums are found, the search is terminated. As finding optimums has lower priority and the focus is to find a parameter set that will yield good results, all the instances have been run for the same amount of time regardless of problem size. The comparison of the results is based on the average objective value for each parameter set.

6.3.2 Tabu tenure

Value of the tabu tenure θ is set to be as Cordeau et al. (2001) proposed, with $\theta = [7.5 log_{10} n]$, both for the standard and time-dependent solver. The value of θ is rounded to the nearest integer.

6.3.3 Infeasibility

A solution can be infeasible if the load of the tours exceed the vehicles capacity. To move the search to other neighborhoods, exploring the infeasible region is allowed. The parameter δ is used to control the behavior when infeasible. Two strategies have been implemented for this.

The parameter α gives the weight of infeasibility in the move evaluation function. The two different strategies implemented concerns the way this parameter is adjusted during the search. Cordeau et al. (2001) propose a geometrical adjustment of α where α is multiplied or divided with the value of δ. For this thesis, δ has been extended with separate values denoted $\delta+$ and $\delta-$. Oppen and Løkketangen (2006) use an arithmetical adjustment where the value of $\delta+$ is added and $\delta-$ is subtracted to/from the current value of α. Using different values for $\delta+$ and $\delta-$ could help the α being more rapidly decreased than increased, meaning that it will guide the search quickly out of the infeasible region.

Test results show that the approach proposed by Cordeau et al. (2001) works best for both solvers. For the CVRP solver, testing also shows that when using two different values for the adjustment of α, the best values for $\delta+$ and $\delta-$ is found to be 1.5 and 5. The best initial value of α has proven to be $\alpha = 10$. For the TDVRP solver these values are $\delta+ = 1.02$, $\delta- = 1.5$ and initial value $\alpha = 5$.

6.3.4 Diversification strategy and parameters

To further explore the solution space a diversification strategy is used to guide the search into other regions of the solution space. For the CVRP solver, the strategy proposed by Cordeau et al. (2001), described in Section 6.1.2, performs best. This yields good results as well as providing the result within a reasonable amount of time. This strategy with the parameter values $\eta = 1.1$ and $\lambda = 5$ will be the ones used for the CVRP solver.

For the TDVRP solver, the diversification strategy proposed by Oppen and Løkketangen (2006) yields the best results and the values $\eta = 1.05$ and $\lambda = 0.015$ will be used.

Results for the preliminary testing can be found in the Appendix C.

6.3.5 Initial solution

Four construction heuristics has been implemented to generate starting solution for the TS. Having a good starting solution could help the tabu search to find good solutions faster. The four heuristics are the construction heuristic proposed by Cordeau et al. (2001), Sweep algorithm by Gillett and Miller (1974), Savings algorithm by Clarke and Wright (1964) and a pure greedy approach where elements are inserted iteratively at places that yield the lowest cost.

As finding an optimal starting solution is not essential for this thesis, only brief testing has been done. The Savings algorithm adapted for the VRP seem to perform well thus this is chosen. Testing also show that although the Savings algorithm use length when constructing tours, this also give the best initial solutions for the TDVRP solver.

7 Computational experiments and results

A summary of the computational experiments and the results obtained are presented in this section. The purpose of this thesis is as mentioned to include ferries in a VRP. As the goal is not to develop superior solvers, but to look at the possibility of inclusion and possible gain of considering ferries, the experiments are done primarily with focus on quality of solutions and computational effort.

7.1 Test cases with ferries

No published test instances with ferries have been found in the literature. To make a comparison to the corresponding instance without ferries, new test instances have been generated. This has been done by modifying a program entitled *Veiviseren* (see Section 7.1.1). The modified version takes a set of locations and provides test instances suitable for the VRP-solvers. In addition, it provides data that can be used by CPLEX (see Appendix A) to solve small instances to optimality.

7.1.1 Veiviseren

As part of a bachelor degree at HiM, Gjendem et al. (2005) made a program called *Veiviseren*. *Veiviseren* takes two or several locations as input and provides shortest path between all locations. The program uses *Elektronisk vegnett - Elveg*, a database that contains all the information on roads in Norway. *Elveg* is combined by two databases where all the geographical data about roads are taken from *Veidatabasen - Vbase* and combined with road information from *Statens Vegvesen - Vegdatabank*. In addition to the geographical road information, this also gives detailed information on speed limits, axle load limitations, height limitations, physical road blocks and so on. *Veiviseren* uses the shortest path algorithm by Dijkstra (1959), with a binary heap as the primary data structure.

7.1.2 Strategy

The goal for this thesis is to compare solution quality and computational effort between a standard VRP solver and an extended solver that include ferries, thus two sets of test instances have to be generated. These have to be identical with the exception of ferries. Modifications has been done to *Veiviseren* in order to get the information needed to make these test instance sets.

7.1.3 Modification of *Veiviseren*

As mentioned, *Veiviseren* provides shortest paths between two or more locations. These paths, or routes, are stored and can be processed after the algorithm has finished. In a normal network topology the link between two nodes would be represented as one edge. As this is calculated from real world data, to travel from i to j could contain a lot of arcs. These have to be traversed to gather the distance of each arc. The distance of all arcs traversed would make the total distance from i to j.

The shortest path algorithm does not consider ferries as it finds the shortest path on the road network in distance, where ferry crossings are given as zero distance. A detour with shorter travel time than the ferry's crossing time will not be considered, thus there can exist alternative routes that have lower travel time. However, this seems highly unlikely on our data sets.

Finding the travel time from i to j is done simultaneously with calculating the travel distance, calculated by dividing the length of the arc with its speed limit multiplied with a speed factor (less than 1), which increases the travel time slightly. The use of a speed factor is done to try to simulate the behavior of larger vehicles accelerating more slowly and not being able to keep the speed on roads with lower standards.

7.1.4 Ferries in test case

Elveg contains all the information about the road network that is needed. As just some of this was included in *Veiviseren*, modification was necessary so that the ferry docks and ferry connections could be identified. As the arcs are traversed, connecting vertices are also checked for if a ferry dock is encountered. If so, and the next arc is a ferry crossing, the route itself is marked as route containing one or more ferries. When a route contains one or more ferries, intermediate times are also stored for the route up until each ferry encountered and from the last ferry to the destination, as mentioned in Section 4. To make the travel time matrix, the travel times for each route containing ferries are then calculated for all time intervals. To find the travel time up until the ferry, the first intermediate time is read. The ferry's timetable is then processed to find the next possible departure and time is set equal to that departure. Finally the ferry's crossing time is added. This is then done for all ferries in the route (if more), and the time from the last ferry to the destination is added.

As mentioned in Section 4, the resolution of T (number of intervals) will heavily affect the size of the file containing the time-dependent travel times. By making spans of intervals, the filesize is reduced. The idea is that as long as the vehicle reach the same ferry from several intervals, the travel time is decreasing proportional to the interval. E.g traveling from interval 0 and 5 connecting to the same ferry using 5 minute interval, the travel time at 5 is the same as for 0 subtracted the interval. This is also valid when there are several ferries as the non-passing property is always satisfied by the last ferry on the arc (i, j). Figure 2 shows how the travel times decreases towards each ferry departure. When departing from an interval not connecting with the ferry, waiting time occurs and the travel time immediately gets larger.

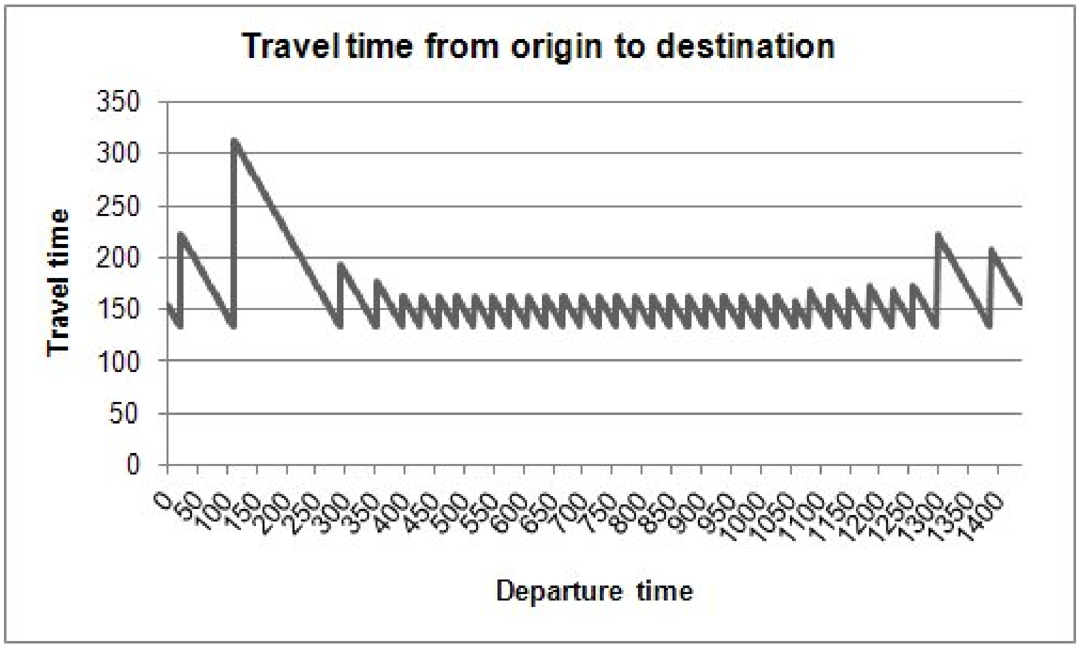


Figure 2: Travel times for a selected O-D pair containing ferries in a 24-hour period. Each low point is a ferry departure.

Figure 3 shows how span of intervals can be used to store the time-dependent travel times. When connecting to the same ferry, departing from origin at different intervals may result in the same arrival time at the destination. This is illustrated as a step function where each step is a ferry departure.

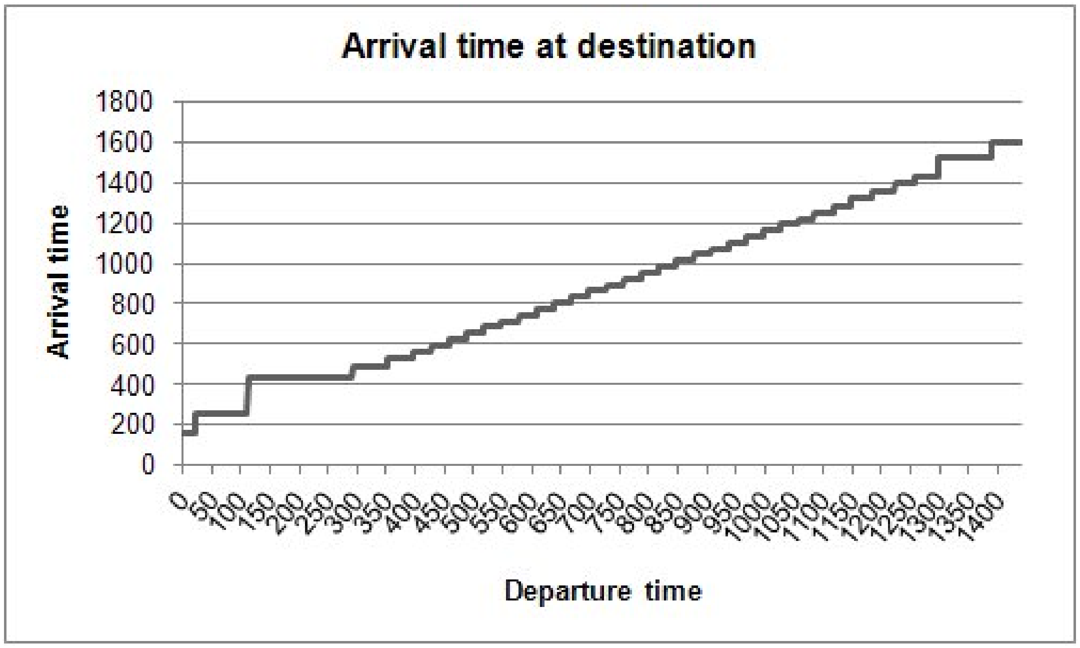
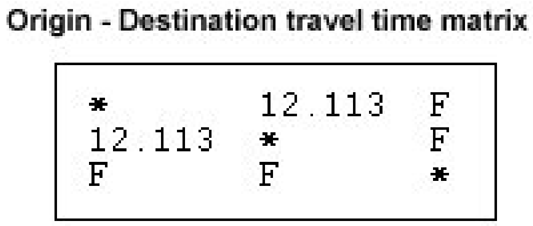


Figure 3: Arrival times at destination when departing from origin at given intervals.

In Section 7.5, test results regarding the resolution of T is presented in terms of generating and reading all intervals as well as span of intervals.

Figure 4 shows output from a small instance containing ferries. The matrix contains static travel times for every (i, j) not containing ferries while others are marked with 'F'. This signals that travel times for this (i, j) is found in the file containing time-dependent travel times. It is worth noting that as the travel times where ferries occur are not symmetrical, as there are entries for both (i, j) and (j, i).

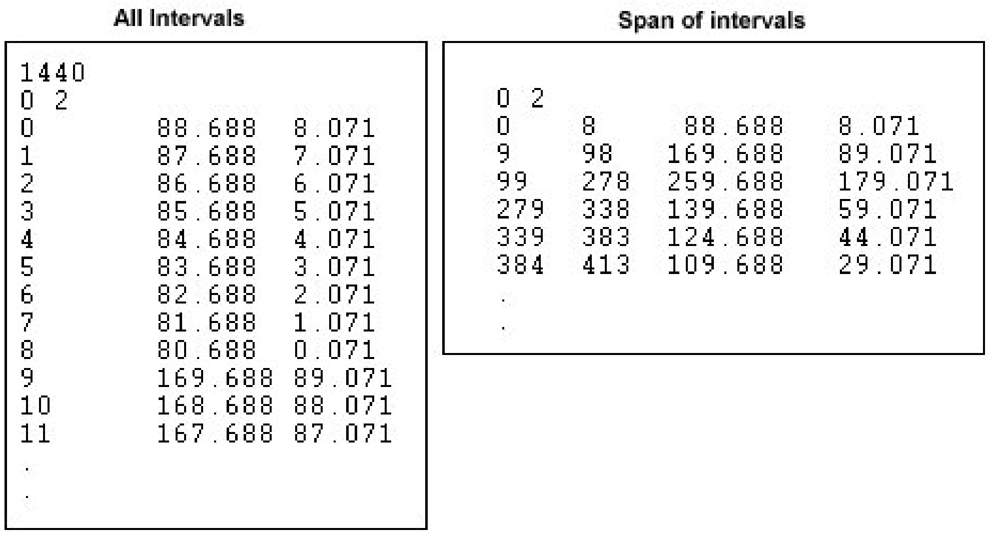

Origin - Destination travel time matrix

```
*       12.113  F
12.113  *       F
F       F       *
```

Figure 4: Example output from *Veiviseren*. Matrix containing entries for ferries and travel times without ferries

Figure 5 shows the output for the time-dependent travel times for routes with ferries.

The figure show how creating spans of intervals save a huge number of entries in the file when p is large. The file containing all intervals starts by giving the number of intervals used (p). For each (i, j) a header with start node i and end node j is printed as well as travel and waiting times for all intervals. In the file containing the spans of intervals the same header with start node i and end node j is given. For each span, the start (S) and end interval of the span is given as well as the travel time (TT) and waiting time (WT) for the first interval. The departure time for the intervals can then be calculated as (TT - (Departure time - S)). Waiting time is calculated using the same procedure as (WT - (Departure time - S)).



All Intervals

```
1440
0 2
0        88.688   8.071
1        87.688   7.071
2        86.688   6.071
3        85.688   5.071
4        84.688   4.071
5        83.688   3.071
6        82.688   2.071
7        81.688   1.071
8        80.688   0.071
9        169.688 89.071
10       168.688 88.071
11       167.688 87.071
.
.
```

Span of intervals

```
0 2
0     8      88.688    8.071
9     98    169.688    89.071
99    278   259.688    179.071
279   338   139.688    59.071
339   383   124.688    44.071
384   413   109.688    29.071
.
.
```

Figure 5: Example output from *Veiviseren*. Departure, time-dependent travel times and waiting time for all intervals on the left. First and last interval of the span, time-dependent travel times and waiting time on the right.

7.2 Cases from real world

Two companies from the local area has been kind enough to provide some of the customer data from their distribution plans. This is used to make proper test cases to simulate real world situations.

7.2.1 Nortura

Nortura is a big company resulting from a fusion of what was formerly Gilde Norsk Kjøtt BA and Prior Norge BA. Nortura is the leading market participant in the meat and egg industry. Nortura has 39 production sites around Norway, including the two sites in Ålesund and Oppdal who has provided customer data for this thesis. This customer data is mainly livestock collection for slaughter houses as used by (Oppen 2008). The data does not contain any customer demands, only locations, so the demands have been generated for own purposes.

7.2.2 Oskar Sylte

Oskar Sylte is a local soft drink producer founded in 1929 and located in Molde, Norway. They deliver to all of Norway although the main customer base is in the county of Møre og Romsdal. The customer data provided by Oskar Sylte is from this county and is for the pickup and delivery problem (VRPPD). As this thesis is focused on ferries in a VRP we only use the customer locations to make the real world instances. Demands are generated as in Section 7.2.1.

7.2.3 Test cases

Test cases have been generated based on real world customer data provided by the two companies, Oskar Sylte and Nortura. The generated instances used for testing are presented in Table 3, with the corresponding tightness ratio and density of ferries. The tightness ratio is the total demand of customers in relation to the total capacity of the available vehicles. A tightness ratio of 1.0 indicates that there are exactly as much demand as there are capacity. The density of ferries is, as explained in Section 2.4, the number of ferries in relation to number of arcs. Also, m is the number of available vehicles while Q is the capacity for each vehicle.

Table 3: Test cases generated from real world customer data.

Instance	m	Q	n	**Tightness ratio**	**Density**
OS-27	4	100	27	0.98	0.03
OS-31	5	100	31	0.89	0.43
OS-58	9	100	58	0.94	0.49
OS-116	18	100	116	0.92	0.65
Nortura-97	15	100	97	0.93	0.74
Nortura-273	50	100	273	0.80	0.73

7.3 Testing environment

To make the work load less and testing more efficient, a small test system has been made. A script takes the parameters and instances from a configuration file and generates tests for all possible combinations of these. These combinations are put into a database system. Here all the combinations are stored as a set of tests with a status field of value 0, 1 or 2. 0 represent a pending test, 1 is while the test is being executed and 2 is when this test is finished. The configuration for the solver then queries the database for a test with status 0. If one is found, this is set to 1 and tested. When the solver is finished it sets the status to 2 and stores all relevant data in a separate table together with the test results.

This setup gives much flexibility. Other parameters can be added while testing is being done. It also helps keeping track of what values are being tested if something should happen during testing, e.g. power failure, system shutdown and so on.

7.4 Validation of the time-dependent solver

As mentioned in Section 7.1 no test cases with ferries have been found in the literature. A small test case with $n = 4$ and ferry density of 0.5 have been generated to manually validate the solver. The results from the solver was then checked manually to make sure the correct travel times where chosen, both in the case of static and time-dependent travel times. In addition, real world data such as the ferry's schedule and the intermediate times to get to and from the ferries where checked.

7.5 Resolution of T

The resolution of T is of great importance to the precision of the time-dependent travel times (see Section 4). A high resolution, a large p, would be beneficial as the precision would get correspondingly better. But increasing p also result in a larger amount of data. As mentioned in Section 7.1.4 this can be avoided by using spans of intervals.

Several tests have been performed to test the times used to generate and read the instances into the solver, as well as the filesize for the time-dependent travel times and memory usage for the TDVRP solver. Both the aspect of generating all p intervals and generating the span of intervals have been tested for $p \in \{1440, 720, 288, 144, 96, 48, 24\}$. All tests have been run 10 times to achieve some statistical significance.

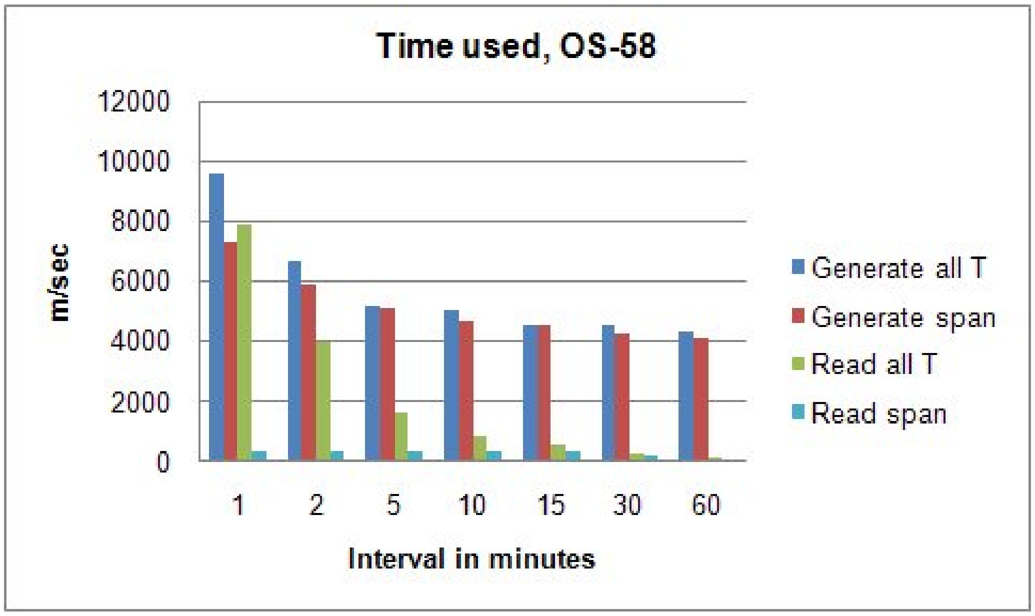


Figure 6: Time usage for generating and reading instance *OS-58*

Figure 6 shows that using span of intervals improves the time usage for generating but most notably for reading the travel times into the solver. In this thesis it is assumed that companies solving VRP's daily do not have substantial changes in the instances often except for customer demands, thus the time used by the solver is of greatest importance. Figure 7, showing the resources used for the same instance, also gives an indication that the approach using span of intervals is preferable.

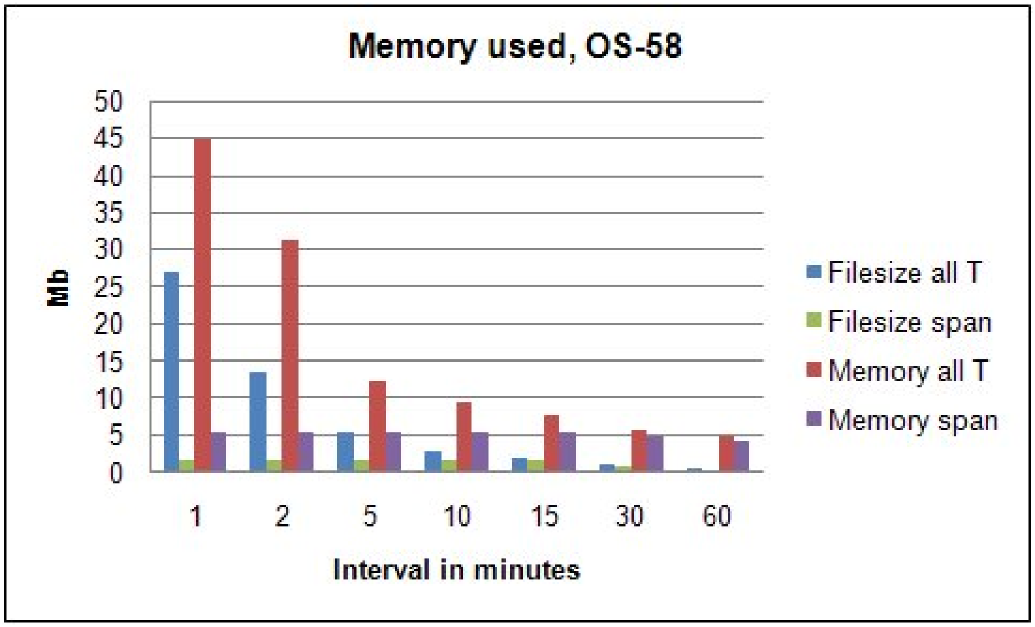


Figure 7: Memory used when generating and reading instance *OS-58*

Figures 8 and 9 shows the same tests performed on a bigger instance, *Nortura-273*. It is apparent that using span of intervals need less time and computer resources for the same resolution of T. Using span provides an opportunity to solve bigger instances using a high resolution of T, as this is dependent on available memory. Faster initiating of the solver is also desirable as many companies need solutions quickly.

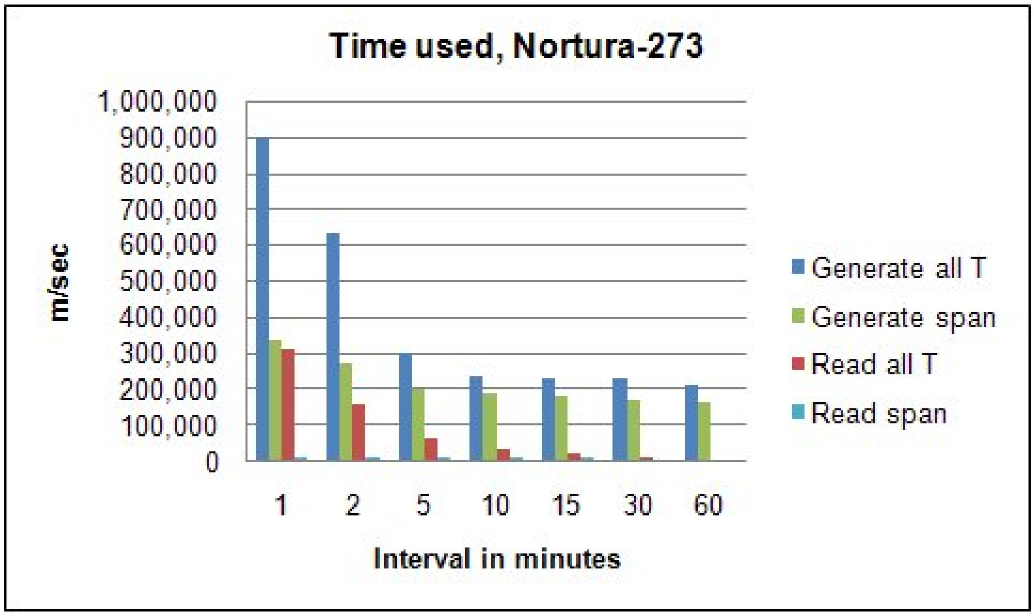


Figure 8: Time usage for generating and reading instance *Nortura-273*

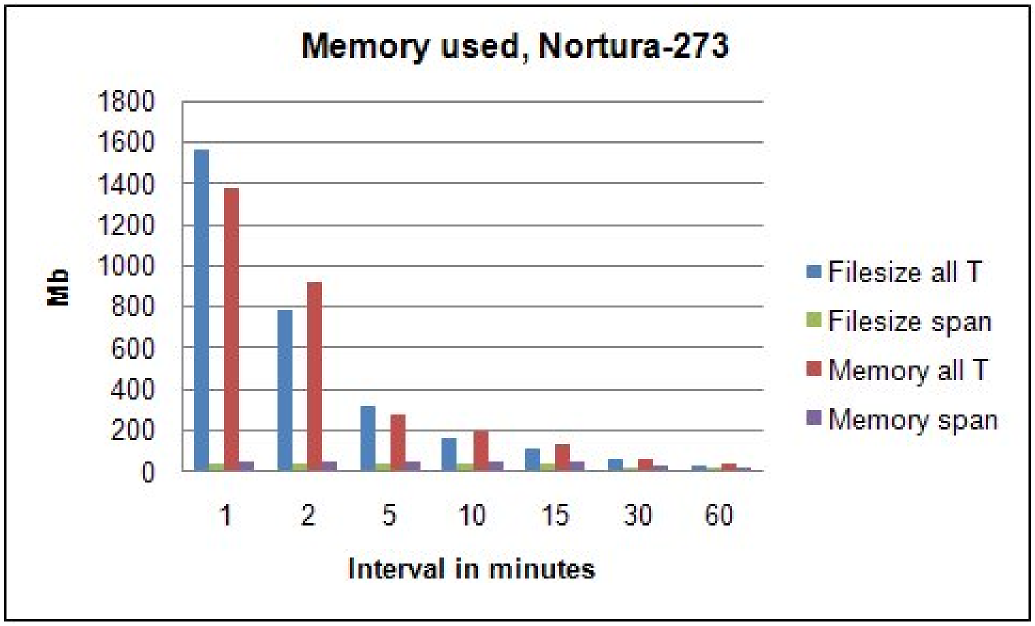


Figure 9: Memory used when generating and reading instance *Nortura-273*

To evaluate if the approach using span of intervals can be used in the solver, it is necessary to test the difference in computational effort in the solver. Reading data and initializing the solver quickly is of little or no use if the computational effort is considerably increased. The same instances used for the tests of the resolution of T has been used to test the difference in computational effort. To achieve statistical significance the tests have also been run 10 times.

Table 4: Average time in m/sec used for 500 iterations having travel times for all intervals or span of intervals and percentage of extra computational effort using span

	All T	**Span**	**Increase**
OS-27	1531	1547	1.05%
OS-31	2094	2266	8.21%
OS-58	9051	9398	3.84%
OS-116	49020	52052	6.19%
Nortura-97	31343	35750	14.06%
Nortura-273	418654	434595	3.81%

Table 4 shows that the computational effort is marginally larger when using span. When the approach having all the intervals have two direct lookups, the approach using span have to find the correct span and then calculate the travel time for the departure time.

Choosing the right resolution for T and the approach to use to store the time-dependent travel times, is very much dependent on what instances to solve and resources available. Also, the strategy of finding the solutions are also of importance. If memory is not a problem and one plan to run the instance for a long time to get a near optimal solution, using the approach of storing the travel times for all intervals would be beneficial. A faster search would compensate for the time to initiate the solver if the run time is long enough. Using span of intervals would lead to faster initializing of the solver, thus providing solutions quicker. Using span would also save a lot of memory, giving the opportunity to have a high resolution of T even for large instances.

As mentioned, a high resolution of T gives better precision. In this thesis a high resolution of T with $p = 1440$ will be used, meaning that travel times will be generated starting at all minutes of the day. To be able to use $p = 1440$ for instances of all sizes, span of intervals will also be used. Looking at the benefits of memory saving and speed of initiating the solver, the decrease in performance is insignificant.

Details for the tests regarding the resolution of T can be found in Appendix D.

7.6 Analysis of solutions from a CVRP solver using the TDVRP solver

As mentioned in the introduction, not considering ferries in a VRP when ferries are in fact present would cause the solutions to be inaccurate. In fact, solving the problems without considering ferries is solving the wrong problem. The TDVRP solver made for this thesis have been extended with an extra feature to illustrate this, having the possibility to read a solution provided by the CVRP solver. The TDVRP solver then traverse the routes provided by the CVRP solver and finds the exact travel times when using ferries for the chosen resolution of T.

7.6.1 Generated test case

For the purpose of showing that not considering ferries in VRP solvers might cause the solution to be infeasible, a test case have been constructed. The instance constructed is of size $n = 8$ and number of vehicles $m = 1$ for simplicity. Only having one vehicle would make the constructed problem a TSP, but as the purpose is to prove the infeasibility of the route this is insignificant. There is one ferry connecting one node to the rest of the nodes in the topology. This ferry has one departure a day in each direction, constructed so that the vehicle have sufficient time to reach the customer and the return departure of the ferry.

Figure 10 show the generated test case in a map of the area. The depot is located in the outer region of the area, while most of the customers are clustered in the city. Customer 5 is disjoint from the rest of the topology and is connected using a ferry connection. As mentioned this ferry is set up having only one departure each day in each direction. The ferry departs at such a time so that the vehicle must go directly to the dock. This means that customer 7 have to be visited on the way back. For simplicity the service time at all customers is 0.

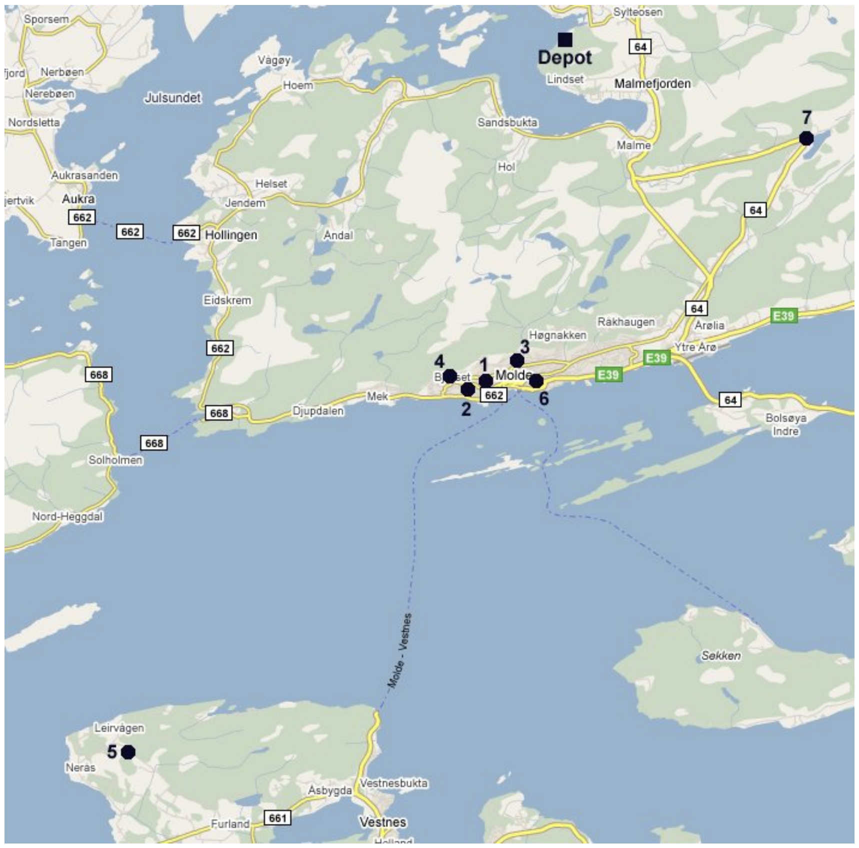


Figure 10: Generated test case shown in map. $n = 8$

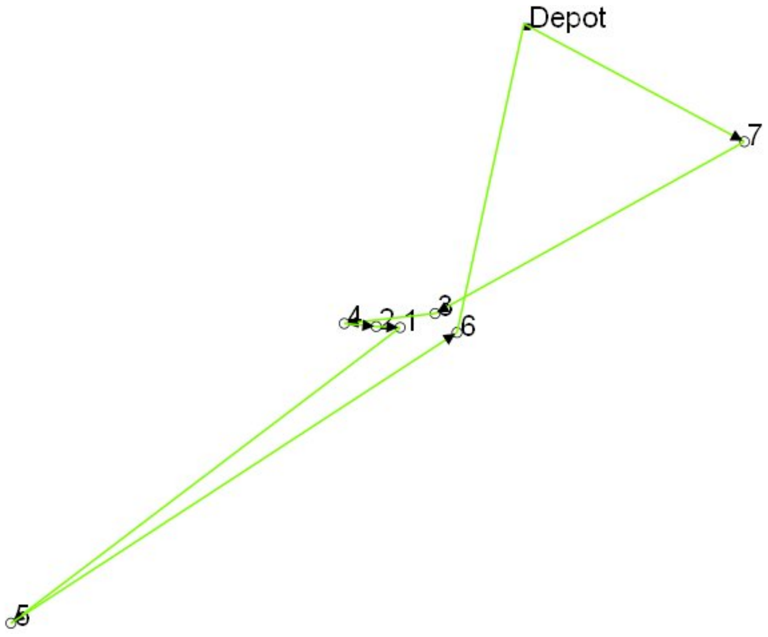


Figure 11: Solution to test case provided by CVRP solver. Duration is 123.75 minutes

The solution to the test case provided by the CVRP solver is shown in Figure 11. Shown in Figure 12 is the solution provided by the TDVRP solver considering ferries. Comparing the two solutions it is apparent that both solvers provides the same sequence of customers, only in opposite directions. As mentioned, if considering ferries, the vehicle need to go directly to the docks. Customer 6 is on the shortest path found from depot to the disjoint customer 5 and service time at all customers are 0, thus the vehicle still reach the ferry departure if customer 6 is visited before embarking the ferry.

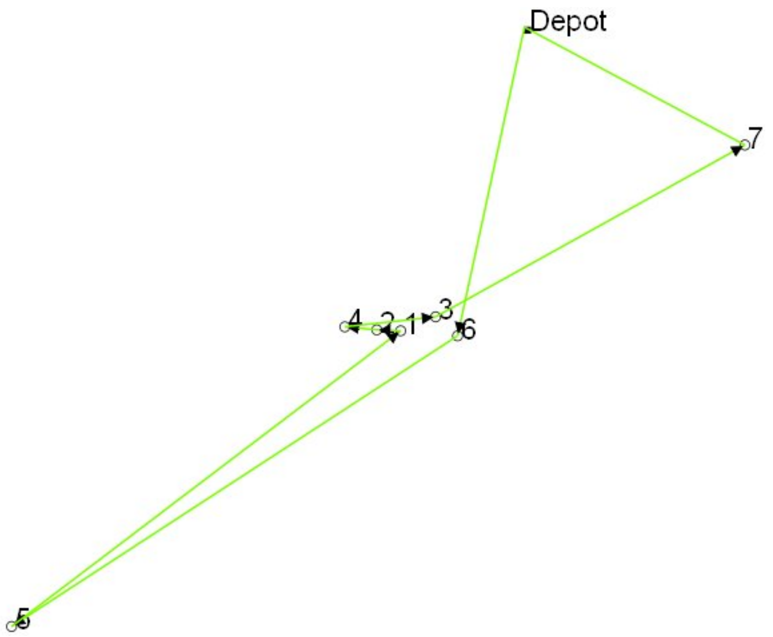


Figure 12: Solution to test case provided by TDVRP solver. Duration is 229.15 minutes

Table 5: Results from generated test case. Shows duration and expected waiting time for CVRP and TDVRP as well as the implemented solution for CVRP.

	Duration	**Waiting**
Planned CVRP	123.75	0.00
Implemented CVRP	1649.12	1455.54
TDVRP	229.15	35.33

Table 5 shows the results from solving the generated test instance. To find the actual duration of the solution provided by the CVRP solver, the solution is implemented in the TDVRP solver. The implementation shows that the actual duration of the solution is 1649.12 minutes, a difference of 1525.37 minutes. As the ferry only departs one time a day, not connecting with the ferry departure result in almost 24 hours of waiting time. The total waiting time for the solution provided by the CVRP solver is 1455.54 minutes compared to 35.33 in the solution provided by the TDVRP solver considering ferries.

Even though this test case is generated, it shows how a solution provided by a CVRP solver might be wrong. The case itself is not unlikely for this region, as there for instance are many farmers living in remote places with ferry connections having few departures a day. For routes not containing many customers, drivers with local knowledge would possibly adjust the route according to the ferry schedule. When the number of stops on the route gets larger, manually adjusting the route would be harder. The manually adjusted route itself would in some cases also be of poor quality compared to a route provided by a solver considering ferries.

7.6.2 Real world test cases

To further look at the quality of solutions provided by the standard solver, all test cases made from the customer data provided by Oskar Sylte (see Section 7.2.2) with $n = \{27, 31, 58, 116\}$ and Nortura (see Section 7.2.1) of size $n = \{97, 273\}$, have been used. The test cases *OS-27* and *OS-31* are based on customer data in urban areas, while *OS-58* also include rural areas. Test case *OS-116* is a larger test case with *OS-27*, *OS-31* and *OS-58* merged. For the instances *Nortura-97* and *Nortura-273*, the customers (farmers) are mostly located in rural areas, but as the customers are spread over a large area, most of the areas road network have to be used. Ferries in urban areas carry more traffic than ferries in rural areas thus they generally have more frequent departures. All tests are run for 50 000 iterations for all sizes and for both the CVRP solver and the TDVRP solver.

Table 6: Results from the real world test cases. Shows duration and waiting time in planned and implemented solution from CVRP solver and solution from TDVRP solver.

	CVRP	**CVRP in TDVRP**		**TDVRP**		
	Duration	Duration	Waiting	Duration	Waiting	Tours
OS-27	847.63	958.25	70.21	909.75	1.86	4
OS-31	219.08	424.60	65.37	420.09	38.88	5
OS-58	928.22	1517.03	419.33	1189.77	51.45	9
OS-116	3484.96	5022.99	597.62	4635.25	212.15	18
Nortura-97	4916.49	6620.79	911.15	6064.39	141.42	15
Nortura-273	13957.40	20629.00	3857.87	17745.90	765.03	42

Table 6 shows the results from the two solvers on the real world test cases. As the CVRP solver does not consider ferries, these values are not directly comparable. Reading the solution into the TDVRP solver would give the answer to what the total time used for the solution would be with ferries. The column "CVRP in TDVRP" show the real value of the CVRP solution implemented in the TDVRP solver. From the results it is apparent that not considering ferries yield a lot of waiting time compared to the solution from the TDVRP. Table 7 shows results from the comparison between planned and implemented solution from CVRP. The extra time that the solution would require is shown, as well as the potential for improvement in solution provided by the CVRP solver. From the results it is apparent that not considering ferries cause solutions to be of poor quality, especially when ferries have less frequent departures.

Table 7: Extra time needed from planned to implemented solution from CVRP solver, as well as potential for improvement compared to the TDVRP solver in time and percentage.

	Extra time CVRP	**Improvement potential**	**Improvement potential**
OS-27	110.63	48.51	5.06%
OS-31	205.52	4.51	1.06%
OS-58	588.81	327.26	21.57%
OS-116	1538.03	387.74	7.72%
Nortura-97	1704.30	556.40	8.40%
Nortura-273	6671.60	2883.10	13.98%

Table 8: Example of a tour in solution from CVRP solver. *OS-58*.

Length in km	**Load**	**# orders**	**Started**	**Finished**	**Duration**	**Waiting**
129.892	99	7	540	1141.310	601.314	355.351

Table 9: Tour details for CVRP solver example tour.

Arrival time	**Customer**	**Demand**	**Location**	**Waiting on arc**
977.377	**42**	**19**	**42-1260166-Sæbø**	**345.240**
977.445	41	15	41-1260174-Sæbø	
1007.800	35	20	35-3055640-Volda	
1013.850	33	8	33-1260539-Volda	
1015.940	30	10	30-1258822-Volda	
1017.800	36	17	36-3007209-Volda	
1029.100	45	10	45-1258228-Hovdebygda	
1141.310	0	0	Depot	10.091

Tables 8 and 9 shows one tour from the solution provided by the CVRP solver. The selected tour is the tour yielding the most waiting time. As shown in the tour details in Table 9, the vehicle going directly from the depot to customer 42 yields a lot of waiting time (illustrated in Figure 13). This is because a ferry connection to this customer has few departures a day. As this waiting occurs to the first customer, the departure could have been delayed from the depot. As the solution is provided by the CVRP solver this is not apparent to a driver. This is because the CVRP solver does not provide any waiting time. It can also be the case that there are one or more ferries prior to the ferry yielding a lot of waiting time. Tables 10 and 11 show the route from the TDVRP solver where the same customer is included. Another sequence of customers is chosen, avoiding the ferry with few departures. As the TDVRP solver use time-dependent travel times, it can also have the possibility to check if there are other departure times from the depot that yield better solutions. As Table 10 shows, moving the departure time from the depot forward with 47 minutes give a better route. Note that moving departure times form the depot is only done for this example and that start of day is set to 540 (09.00) as default.

Table 10: Example of tour in solution from TDVRP solver. *OS-58*

Length in km	**Load**	**# orders**	**Started**	**Finished**	**Duration**	**Waiting**
170.803	99	8	493	781.314	288.314	1.272

Table 11: Tour details for TDVRP solver example tour.

Arrival time	**Customer**	**Demand**	**Location**	**Waiting on arc**
591.833	43	6	43-1280008-Ørsta	0.686
599.772	45	10	45-1258228-Hovdebygda	
610.725	28	6	28-3028316-Volda	
611.274	34	11	34-3042830-Volda	
612.894	30	10	30-1258822-Volda	
648.276	41	15	41-1260174-Sæbø	
648.344	**42**	**19**	**42-1260166-Sæbø**	
684.737	47	22	47-1212597-Ørsta	
781.314	0	0	Depot	0.586

Figure 13 show the area of the routes presented in Tables 8 to 11. The area is represented as a real world map with the selection of customers from the two routes plotted. As the TDVRP solver optimize the routes with ferries considered, the TDVRP solver choose a route going west on the main roads, avoiding the ferry with few departures marked with an *A* in the map. The CVRP solver chooses the shortest path going south to a more remote area thus using the mentioned ferry.

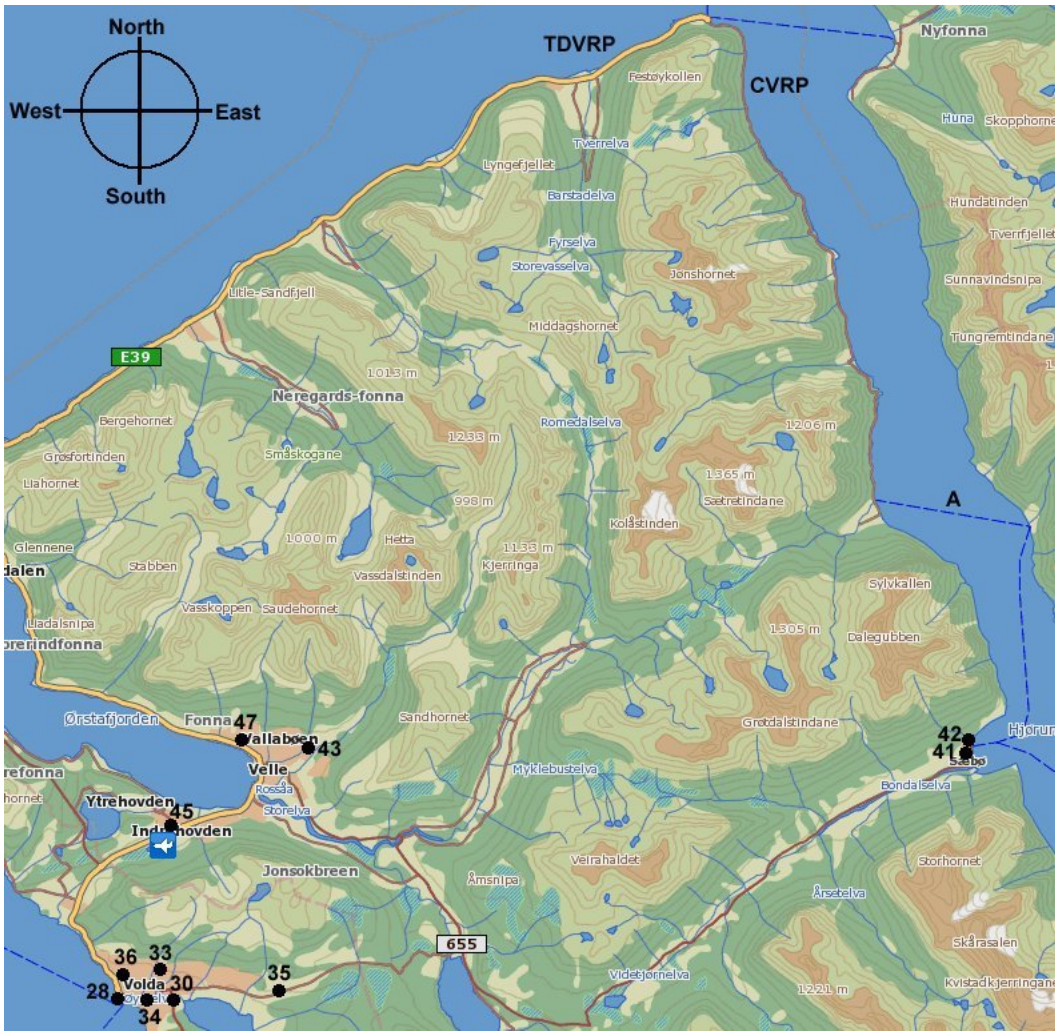


Figure 13: Illustration of the region of customer 42 in test case *OS-58*

7.7 Using approximation to consider ferries in VRP

Approximation is one way to include ferries in a VRP. By using an estimate of the travel time where ferries occur as a static travel time, approximating the time used is possible. Approximation would give the opportunity to use a CVRP solver to handle ferries in a VRP, without the need of a solver using time-dependent travel times. This thesis consider two approaches to approximation, both are approaches that can be implemented with little effort.

To compare the approximation approach with the TDVRP solver, approximation instances have been made for the same instances as used in Section 7.6.2. The instances are identical with the exception of the travel time on arcs (i, j) where ferries occur. The idea is to compare the routes suggested by the solver as well as the total time used, to see if approximation is an acceptable approach toward ferries.

7.7.1 Approximation on sailing time

One approximation approach that can be used, is an approximation based on the sailing time of the ferry. The departures of the ferry are often based on the sailing time of the ferry, as they run continuously. For example, if one ferry serves a ferry connection, the departure times are based on the sailing time added enough time to reload the ferry. The first approximation tested in this thesis is based on the sailing time with half the sailing time added as average waiting time. This is a fair approach towards ferry crossing with frequent departures. When generating instances, the only extra information needed is the sailing time of the ferry.

Table 12 show the planned results of the approximation approach compared to the planned results from the CVRP and TDVRP solvers. From the table it is apparent that the planned solution based on the approximation uses more time. This is natural as the CVRP solver does not have a cost on using ferries.

Table 12: Planned results with approximation approach using sailing times, compared to planned solutions from CVRP and TDVRP solver.

	CVRP	**Approximation**	**TDVRP**	
	Duration	Duration	Duration	Waiting
OS-27	847.63	907.63	909.75	1.86
OS-31	219.08	429.16	420.09	38.88
OS-58	928.22	1178.03	1189.77	51.45
OS-116	3484.96	4787.18	4635.25	212.15
Nortura-97	4916.49	6102.61	6064.39	141.42
Nortura-273	13957.40	17807.80	17745.90	765.03

To see the real effect of the approximation, the planned solution need to be implemented using the TDVRP solver. Table 13 show the planned solutions from the CVRP solver and the approximation implemented in the TDVRP solver compared to the solution provided by the TDVRP solver. The results show that using the approximation based on sailing time only provide significantly better results than the CVRP solver on the largest instance with $n = 273$.

Table 13: Results from CVRP solver and approximation using sailing time implemented in the TDVRP solver, compared to result from TDVRP solver.

	CVRP		**Approximation**		**TDVRP**	
	Duration	Waiting	Duration	Waiting	Duration	Waiting
OS-27	958.25	70.21	958.25	70.21	909.75	1.86
OS-31	424.60	65.37	425.59	67.18	420.09	38.88
OS-58	1517.03	419.33	1547.27	454.53	1189.77	51.45
OS-116	5022.99	597.62	4876.13	526.01	4635.25	212.15
Nortura-97	6620.79	911.15	6606.35	873.56	6064.39	141.42
Nortura-273	20629.00	3857.87	19255.80	2689.72	17745.90	765.03

Table 14 show how the approximation perform in comparison to the implemented solution from the CVRP solver as well as the improvement potential that are in comparison to the TDVRP solver. As mentioned, the approach does not provide substantial improvements over CVRP with the exception of for instance *Nortura-273*.

Table 14: Comparison of using approximation on sailing time to CVRP and TDVRP solver.

	Improvement from CVRP	**Improvement potential**
OS-27	0.00%	5.06%
OS-31	-0.24%	1.29%
OS-58	-1.99%	23.11%
OS-116	2.92%	4.94%
Nortura-97	0.22%	8.20%
Nortura-273	6.66%	7.84%

As the approximation uses the sailing time on the ferry with an estimated waiting time, this would in reality mean that the arc has a cost in relation to the distance. The solver would then consider this link as a normal road with a given cost choosing another sequence of customers if it yields better results. The approach will not help the solver optimize routes to ferry departures, nor will it help the solver avoid ferry connections with few departures.

7.7.2 Approximation on frequency of departures

Another approximation approach is an approach based on the frequency of departures for the ferry. As some ferries could have less frequent departures, basing the approximation on this frequency would give a better estimate of the average waiting time when using this ferry. The second approach tested in this thesis is based on the number of departures during a time horizon similar to a normal work day. For instance, if the time horizon is given as 09.00 (540) to 17.00 (1020), the ferry has 10 departures during this time horizon and the sailing time is 30 minutes. The approximation is then the estimated waiting time added the sailing time, as shown in (19). Waiting time is calculated as the duration of time horizon divided by the number of departures.

$$t_{app} = \frac{end - start}{\#departures} + sailingtime = \frac{1020 - 540}{10} + 30 \tag{19}$$

When generating instances, the number of departures and sailing time of the ferries as well as a chosen time horizon are needed. The time horizon could be the entire day, but as ferries have less frequent departure during the night, a time horizon similar to a normal work day is recommended.

Table 15 shows the planned results from the approximation based on frequency of departures on the same real world test cases used in Section 7.6.2 and 7.7.1.

Table 15: Planned results from approximation approach using departure frequency, compared to planned solutions from CVRP and TDVRP solvers.

	CVRP	**Approximation**	**TDVRP**	
	Duration	Duration	Duration	Waiting
OS-27	847.63	988.63	909.75	1.86
OS-31	219.08	475.16	420.09	38.88
OS-58	928.22	1347.18	1189.77	51.45
OS-116	3484.96	5175.60	4635.25	212.15
Nortura-97	4916.49	6817.80	6064.39	141.42
Nortura-273	13957.40	21442.30	17745.90	765.03

Table 16 show the approximation using the frequency of the ferry combined with the sailing time implemented in the TDVRP solver. The results show that this approximation yield far better results than the approximation based on sailing time on most of the instances. The approach helps to avoid ferries that yield much waiting time thus the result for the instance *OS-58* is significantly better than the result provided by the CVRP solver.

Table 16: Results from CVRP solver and approximation using frequency implemented in the TDVRP solver, compared to result from TDVRP solver.

	CVRP		**Approximation**		**TDVRP**	
	Duration	Waiting	Duration	Waiting	Duration	Waiting
OS-27	958.25	70.21	958.25	70.21	909.75	1.86
OS-31	424.60	65.37	425.59	67.18	420.09	38.88
OS-58	1517.03	419.33	1248.71	109.46	1189.77	51.45
OS-116	5022.99	597.62	4757.85	371.23	4635.25	212.15
Nortura-97	6620.79	911.15	6458.83	589.90	6064.39	141.42
Nortura-273	20629.00	3857.87	20851.20	3828.05	17745.90	765.03

Table 17 is similar to Table 14 for the approximation approach using departure frequency. Results show that the approach helps when ferries have less frequent departures (as in *OS-58* and *OS-116*), but only provide slightly better or even worse results than the CVRP solver.

Table 17: Comparison of approximation using departure frequency to CVRP and TDVRP solvers.

	Improvement from CVRP	**Improvement potential**
OS-27	0.00%	5.06%
OS-31	-0.24%	1.29%
OS-58	17.69%	4.72%
OS-116	5.28%	2.58%
Nortura-97	2.45%	6.11%
Nortura-273	-1.08%	14.89%

It is important to note that even though the approximation yield fairly good results to some of the instances tested in this thesis, using time-dependent travel times is a better approach. As mentioned, the approximation using frequency of departures helps the solver to avoid ferry connections with less frequent departures. If this ferry connection has to be used to reach a customer, the approximation does not optimize the routes to connect with the departure as the solver using time-dependent travel times do.

7.7.3 Effect of ferry density

Testing the effect of ferry density in the instances is very difficult as there are more factors to be considered. The frequency of departures is the most important factor as most of the problems with not considering ferries in a VRP, are related to the waiting time. One could imagine that if all ferries in a topology had a departure every minute, a density of 1.0 with all origin-destination pair connected by one or more ferries would not cause any problems. The solution value would be inaccurate but the routes would be fairly approximate. As the approximation in Section 7.7.1 showed, giving a cost to the ferry without considering departure times is of little or no value, especially when ferries have less frequent departures.

7.8 Computational effort

As mentioned, using time-dependent travel times in the solver will give better solution quality. Just as important is it to get the solutions in reasonable time. To measure the performance, the TDVRP solver has been tested up against the performance of the CVRP solver.

7.8.1 Comparison of executable speed using different compilers

As the C++ compiler from Microsoft, Visual C++ 2008, is restricted to the Windows operating system, it was interesting to see if using a different compiler could have any effect on computational effort. The comparison to Microsoft Visual C++ 2008 has been done using MinGW 5.4.1 ("Minimalistic GNU for Windows"), an open source command-line compiler for Windows based on the GNU GCC project. The two compilers have been tested on number of iterations performed in 10 minutes. Test results show that choosing what compiler to use carefully would be beneficial.

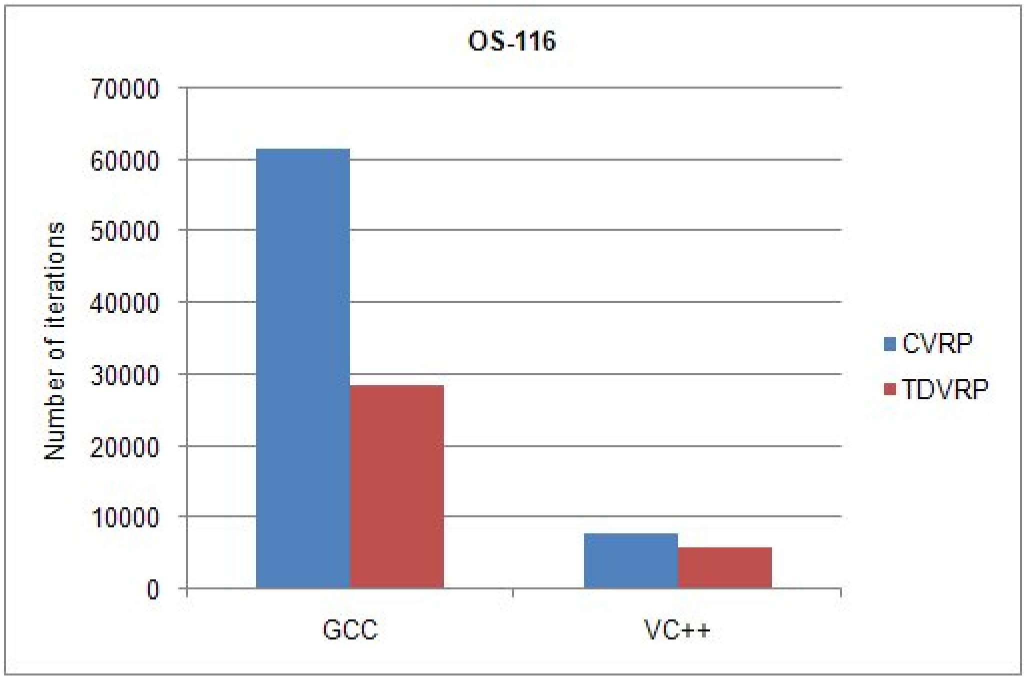


Figure 14: Iteration comparison GCC and VC++

Figures 14 and 15 show the number of iterations performed by the GCC and VC++ compiler for the instances *OS-116* and *Nortura-273*. Results show that the GCC compiler outperforms the VC++ compiler, with the iteration count being around 8 times larger for the CVRP solver and around 5 times larger for the TDVRP solver. This would be of great importance as more iterations in the same amount of time would mean that the solution space is being explored to a greater extent. In addition, the difference in computational effort between the CVRP and TDVRP solver is much larger with the GCC compiler where the TDVRP solver only manage around 50% compared to the CVRP solver. Using the VC++ compiler, the TDVRP solver manage around 80% in comparison to the CVRP solver.

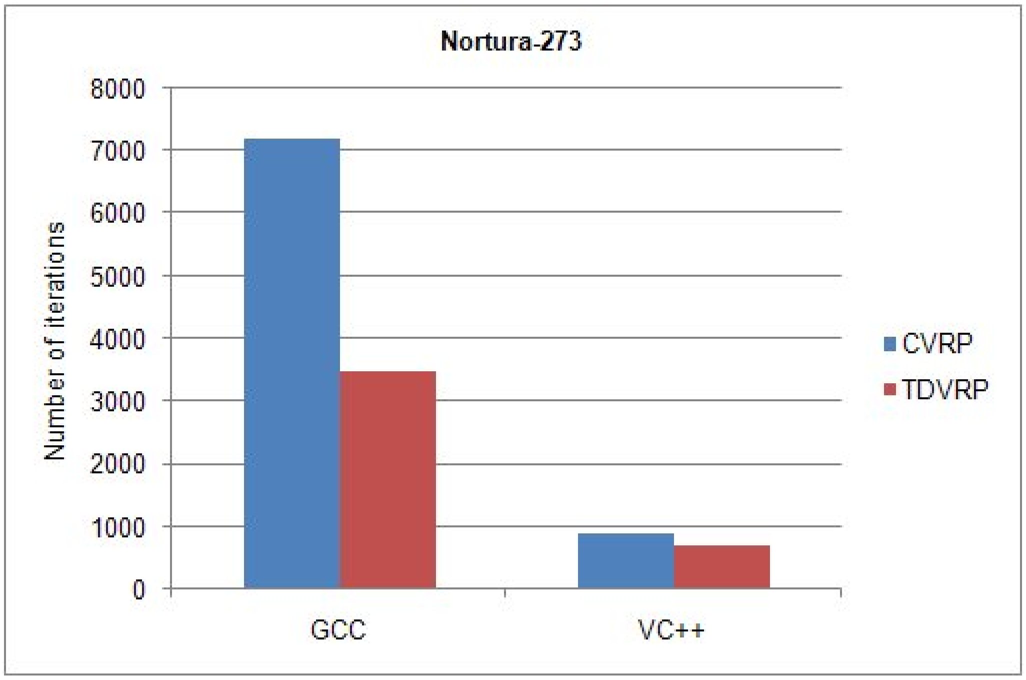


Figure 15: Iteration comparison GCC and VC++

7.8.2 Performance of solvers

In order to compare the performance of the TDVRP solver to the CVRP solver, the instances presented in Section 7.2.3 are used. Each instance is run for 1 hour, for both the CVRP and TDVRP solver. Presented in Table 18 are the time used per iteration for all runs. For each instance, the numbers are calculated from the total time used divided by the total number of iterations done. In addition, the table shows the percentage of the extra computational effort needed for the TDVRP solver. The extra time usage needed vary from 18.44% to 34.43% and the average is 24.28%. As mentioned, the instance *OS-27* is made from customer data in an urban area with low density of ferries, thus the extra computational effort needed for the TDVRP solver is small. In comparison, the results using MinGW GCC compiler are presented in Table 19. Using the GCC compiler, the extra time usage needed vary from 41.60% to 113.14% and the average is 81.01%.

The results show that the choice of compiler is of great importance. Not only is the MinGW compiler a lot faster, the difference in using a TDVRP solver in comparison to a CVRP solver is also much larger than for the VC++ compiler.

Table 18: Time used per iteration using CVRP and TDVRP solvers with VC++ compiler and percentage of extra computational effort using TDVRP solver.

Instance	**m/sec CVRP**	**m/sec TDVRP**	**Extra effort**
OS-27	2.625	3.109	18.44%
OS-31	3.61	4.375	21.19%
OS-58	15.281	18.938	23.93%
OS-116	79.203	100.984	27.50%
Nortura-97	52.094	70.031	34.43%
Nortura-273	759.828	913.046	20.16%
Average			**24.28%**

Table 19: Time used per iteration using CVRP and TDVRP solvers with GCC compiler and percentage of extra computational effort using TDVRP solver.

Instance	**m/sec CVRP**	**m/sec TDVRP**	**Extra effort**
OS-27	0.375	0.531	41.60%
OS-31	0.485	0.797	64.33%
OS-58	1.875	3.453	84.16%
OS-116	9.031	18.297	102.60%
Nortura-97	5.938	12.656	113.14%
Nortura-273	88.844	160.11	80.21%
Average			**81.01%**

For interested readers, details about the performance of the TDVRP solver can be found in Appendix E.

8 Conclusions

The goal for the work presented in this thesis was to include ferries in a VRP solver and look at advantages and the extra computational effort needed. To include time-dependent travel times for origin-destination pairs that contain one or more ferries, an approach using travel times for all intervals of a chosen time resolution is used. Also, an approach using span of intervals to store time-dependent travel times in order to save memory is introduced. As mentioned in the introduction, we wanted to show that considering ferries in a VRP where ferries are in fact present, is beneficial and in fact necessary.

To make instances containing ferries, modification have been done to an existing program entitled *Veiviseren*. Using real world road information and customer data provided by local companies, real world based test cases containing ferries have been generated. Two solvers, a CVRP and a TDVRP solver, using the tabu search metaheuristic have been made to solve the instances and to provide solutions that were comparable.

Test results show that not considering ferries in a VRP when ferries are present, does not provide good solutions. In fact, not considering ferries in a VRP where ferries are present, is solving the wrong problem. By using time-dependent travel times stored in span of intervals, considering ferries can be done in an efficient manner. Results show that considering ferries yield substantial improvements in comparison to a standard CVRP solver using static travel times. Extra computational effort is needed for a TDVRP solver considering ferries, but it is recommended, as a TDVRP solver provides better solutions in terms of total travel and waiting time, and the fact that a TDVRP solver solves the real problem.

8.1 Future work

While working on this thesis, several issues emerged that we found interesting. As a result, there are some issues we would like to point out as an elongation of this thesis. The main focus of this thesis was on the inclusion of ferries in a VRP solver, not to develop a superior solver. As a consequence, there are also several possible improvements we would have liked to have tested.

The tabu status used for this thesis used the combination of customer and tour. Testing different approaches to the tabu status would be interesting to see if this has any effect when using a time-dependent solver. Having the tabu status only on the customer would give a more strict tabu status. As this thesis uses time-dependent travel times for many intervals, it would be interesting to see if a tabu status that also consider the time of day the customer was inserted in the route, would have any effect.

The same issues arise when considering the frequency penalty. Having the frequency updated for only the customer or in a combination with time of day would be interesting.

For the TDVRP solver it would be interesting to do more thorough testing to find the best construction algorithm and the most suitable choice of move. As this thesis is mostly based on ideas towards approaches for tabu search using static travel times, testing if other methods work better for a time-dependent solver could prove beneficial.

In this thesis, the objective is to minimize time. As this is only a sub-problem in the real world, cost-optimizing the routes would be more sensible. Gathering real world data such as driver cost, truck cost, cost per kilometer and so on, from companies would give the possibility to cost-optimize in a reasonable fashion.

Testing the approach in this thesis towards other time-dependent problems in the VRP is another issue for elongation of this thesis. Using the approach with intervals containing the time-dependent travel times could also be used for rush hour, as long as the non-passing property is not violated.

References

Bräysy, O. and M. Gendreau (2001, 12). Tabu search heuristics for the vehicle routing problem with time windows. Technical report, SINTEF Applied Mathematics.

Clarke, G. and J. W. Wright (1964, 8). Scheduling of vehicles from a central depot to a number of delivery points. *Operations Research 12*(4), 568–581.

Cordeau, J.-F., G. Laporte, and A. Mercier (2001, 8). A unified tabu search heuristic for vehicle routing problems with time windows. *Journal of Operational Research Society 52*(8), 928–936.

Dijkstra, E. (1959). A note on two problems in connection with graphs. *Numerische Mathematik 1*, 269.

Donati, A. V., R. Montemanni, N. Casagrande, A. E. Rizzoli, and L. M. Gambardella (2008). Time dependent vehicle routing problem with a multi ant colony system. *European Journal of Operational Research 185*(3), 1174 – 1191.

Donati, A. V., R. Montemanni, L. M. Gambardella, and A. E. Rizzoli (2003). Integration of a robust shortest path algorithm with a time dependent vehicle routing model and applications. In *Proceedings of CIMSA 2003- International Symposium on Computational Intelligence for Measurement Systems and Applications*.

Gendreau, J.-F. and G. Laporte (2005). Tabu search heuristics for the vehicle routing problem. *Metaheuristic Optimization via Memory and Evolution 30*, 145–163.

Gendreau, M., A. Hertz, and G. Laporte (1994). A tabu search heuristic for the vehicle routing problem. *Management Science 40*, 1276–1290.

Gillett, B. E. and L. R. Miller (1974, 4). A heuristic algorithm for the vehicle-dispatch problem. *Operations Research 22*(2), 340–349.

Gjendem, A., E. Berg, and L. Østby (2005). Ruteberegning i det norske veinett. Technical report, Molde University College.

Glover, F. and M. Laguna (1997). *Tabu Search*. Norwell, MA, USA: Kluwer Academic Publishers.

Haghani, A. and S. Jung (2005). A dynamic vehicle routing problem with time-dependent travel times. *Comput. Oper. Res. 32*(11), 2959–2986.

Holland, J. H. (1975). *Adaptation in natural and artificial systems*. University of Michigan, Ann Arbor.

Ichoua, S., M. Gendreau, and J.-Y. Potvin (2003). Vehicle dispatching with time-dependent travel times. *Elsevier 144*, 379–396.

Kerbache, L. and T. van Woensel (2004, December). Planning and scheduling transportation vehicle fleet in a congested traffic environment. Les Cahiers de Recherche 803, Groupe HEC.

Kernighan, B. W. and S. Lin (1973). An effective heuristic algorithm for the traveling-salesman problem. *Operations Research 21*, 498–516.

Laporte, G. (2007). What you should know about the vehicle routing problem. *Naval Research Logistics 54*, 811–819.

Laporte, G. and Y. Nobert (1997). Exact algorithms for the vehicle routing problem. *Surveys in Combinatorial Optimization, North-Holland, Amsterdam 132*, 147–184.

Malandraki, C. and M. S. Daskin (1992). Time dependent vehicle routing problems: Formulations, properties and heuristic algorithms. *Transportation Science 26*(3), 185–200.

Mester, D. and O. Bräysy (2007). Active-guided evolution strategies for large-scale capacitated vehicle routing problems. *Comput. Oper. Res. 34*(10), 2964–2975.

Miller, C. E., A. W. Tucker, and R. A. Zemlin (1960). Integer programming formulation of traveling salesman problems. *J. ACM 7*(4), 326–329.

Nagata, Y. (2007). Edge assembly crossover for the capacitated vehicle routing problem. *Lecture Notes in Computer Science 4446/2007*, 142–153.

Oppen, J. (2008). *Models and Solutions for Rich Transportation Problems.* Ph. D. thesis, Molde University College.

Oppen, J. and A. Løkketangen (2006). Arc routing in a node routing enviroment. *Computers and Operations Research 33*, 1033–1055.

Oppen, J. and A. Lokketangen (2008). A tabu search approach for the livestock collection problem. *Comput. Oper. Res. 35*(10), 3213–3229.

Taillard, D. (1993). Paralell iterative search methods for vehicle routing problem. *Networks 23*(8), 661–673.

Toth, P. and D. Vigo (2002). *The Vehicle Routing Problem.* Society for Industrial & Applied Mathematics.

Appendix A Tools used for this thesis

Veiviseren

Veiviseren is a program that provides the shortest path between locations using a real-world road network.

Microsoft Visual Studio

Development tool by Microsoft with editor and debugger.
`http://msdn.microsoft.com/en-us/vstudio/default.aspx`

Eclipse

Eclipse is a multi-language software development platform comprising an integrated development platfrom and a plug-in system to extend it.
`http://www.eclipse.org/`

PostgreSQL

PostgreSQL is a open-source object-relational database management system.
`http://www.postgresql.org/`

Subversion

An open-source revision control system.
`http://subversion.tigris.org/`

Python

Python is an object oriented programming language.
`http://www.python.org/`

SPSS

SPSS is a powerful statistical analysis program used for doing advanced statistical analysis of data for research and other projects.
`http://www.spss.com/`

TeXnicCenter

LaTeX editor to make it easier to write LaTeX documents.
`http://www.texniccenter.org/`

LaTeX

LaTeX is a document markup language and document preparation system for TeX typesetting program.
`http://www.latex-project.org/`

JabRef

JabRef is a front-end to BibTeX and other bibliographies, used to manage references.
`http://jabref.sourceforge.net/`

AMPL

Modeling language and system for formulating, solving and analyzing large-scale optimization (mathematical programming) problems.
`http://www.ampl.com/`

ILOG CPLEX

CPLEX is an optimization software package, used to solve instances provided in AMPL.
`http://www.ilog.com/products/cplex/`

Excel2Latex

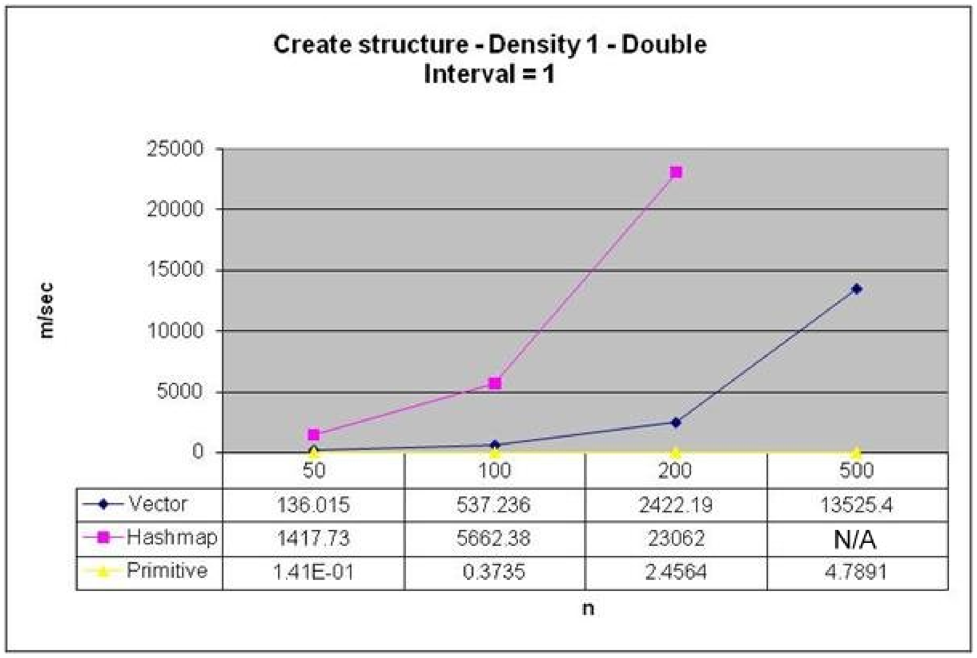
Convert Excel table to LaTeX table format.
`ftp://cam.ctan.org/tex-archive/support/excel2latex.zip`

Appendix B Test of data structures

Test of data structures to store travel times between origin and destination. Tested data structures are Vector, Hashmap and 2-dimensional vector with one lookup presented as primitive structure. In addition, both using double and integer where tested, both showing the similar trends with marginal differences. Shown in Figures 16 and 17 are selected results from creating data structure and lookup time. As the trends are similar for all sizes and density, only density = 1 with double are shown.

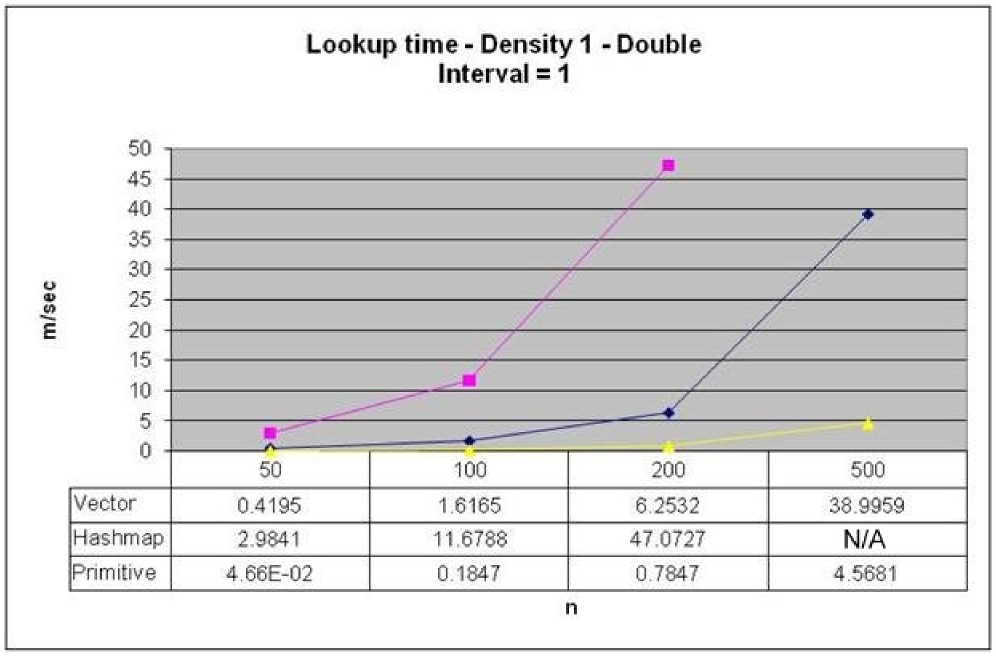
Testing of different densities show that the time to create data structure and time for lookup, grow as a linear function. Density 0.5 have approximately half the times as density 1.0. Test results also suggest that the computational effort grows quadratic in relation to the instance size. This is also indicated by Figure 18, where a quadratic curve is fitted to the observed data (create structure - Vector).

Tests are run on Intel®Core^{TM}2 Duo 2.4GHz with 4GB RAM, using GCC 4.0.1 compiler. All times are shown in milliseconds.



	50	100	200	500
Vector	136.015	537.236	2422.19	13525.4
Hashmap	1417.73	5662.38	23062	N/A
Primitive	1.41E-01	0.3735	2.4564	4.7891

Figure 16: Time to create data structure using 1 minute interval and density 1.0



	50	100	200	500
Vector	0.4195	1.6165	6.2532	38.9959
Hashmap	2.9841	11.6788	47.0727	N/A
Primitive	4.66E-02	0.1847	0.7847	4.5681

Figure 17: Lookup time using 1 minute interval and density 1.0

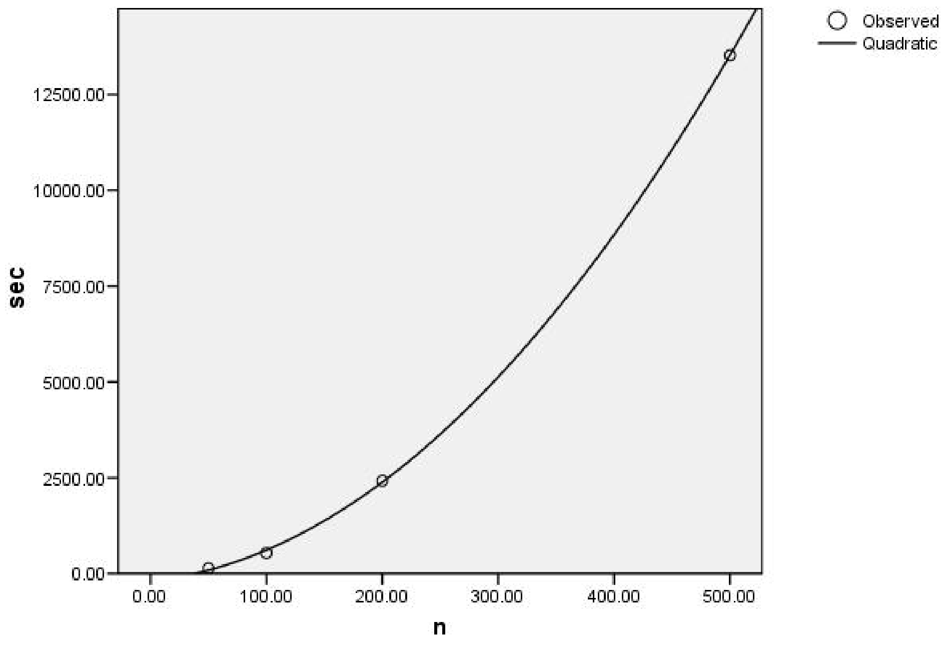


Figure 18: Quadratic curve fittet to the observed data

Appendix C Preliminary testing

To find suitable parameters for the tabu search heuristic, preliminary testing have been done. Parameters for the CVRP solver have been found by trying several parameter sets on a set of instances from the literature, A-n32-k5.vrp, B-n50-k7.vrp, E-n101-k14.vrp, F-n135-k7.vrp and P-n70-k10.vrp. Preliminary testing for the TDVRP solver have been done on the real world cases generated for this thesis.

In the Tables 20 and 21, the diversification mechanism by Oppen and Løkketangen (2006) is represented as DivMode equal to 1, DivMode 2 represent the diversification mechanism by Cordeau et al. (2001). The infeasibility strategy proposed by Cordeau et al. (2001) is represented as DeltaMode equal to 1.

Table 20: Selection of the 20 best parameter sets found for the CVRP sovler, sorted by average deviation from best known values. The selected parameter set is highlighted.

DivMode	α	η	λ	**DeltaMode**	$\delta+$	$\delta-$	**Average**
2	**10**	**1.1**	**5**	**0**	**1.5**	**5**	**0.00195**
2	10	1.1	10	0	1.5	5	0.00195
2	1	1.01	5	0	1.5	5	0.00260
2	1	1.01	10	0	1.5	5	0.00282
1	10	1.1	0.015	0	1.5	5	0.00282
2	10	1.05	5	0	1.5	5	0.00304
2	10	1.05	10	0	1.5	5	0.00304
1	1	1.01	0.015	0	1.5	5	0.00347
2	1	1.1	10	0	1.5	5	0.00347
2	1	1.1	5	0	1.5	5	0.00347
2	1	1.05	10	0	1.5	5	0.00369
2	1	1.05	5	0	1.5	5	0.00369
2	5	1.01	5	0	1.5	5	0.00413
1	10	1.05	0.015	0	1.5	5	0.00413
2	1	1.1	0.015	0	1.5	5	0.00413
2	5	1.01	10	0	1.5	5	0.00434
2	5	1.1	0.015	0	1.5	5	0.00478
2	1	1.05	0.015	0	1.5	5	0.00521
2	1	1.1	5	0	1.5	1.5	0.00543
2	1	1.1	10	0	1.5	1.5	0.00543

Table 21: Selection of the 20 best parameter sets found for the TDVRP solver, sorted by average deviation from best known values. The selected parameter set is highlighted.

DivMode	α	η	λ	**DeltaMode**	$\delta+$	$\delta-$	**Average**
1	**5**	**1.05**	**0.015**	**0**	**1.02**	**1.5**	**0.0136**
1	5	1.01	0.015	0	1.02	1.5	0.0139
1	5	1.05	0.015	0	1.02	5	0.0144
1	5	1.1	0.015	0	1.02	1.5	0.0146
1	5	1.01	0.015	0	1.02	5	0.0172
1	5	1.1	0.015	0	1.02	5	0.0180
1	10	1.05	0.015	0	1.02	5	0.0190
1	10	1.1	0.015	0	1.02	5	0.0190
2	1	1.01	10	0	1.5	5	0.0214
1	10	1.01	0.015	0	1.02	5	0.0222
1	5	1.01	5	0	1.02	1.5	0.0228
2	5	1.1	0.015	0	1.5	1.5	0.0231
1	5	1.1	0.015	0	1.5	5	0.0234
1	5	1.01	10	0	1.02	1.5	0.0239
1	5	1.05	0.015	0	1.5	5	0.0240
1	5	1.05	10	0	1.02	1.5	0.0242
1	5	1.1	10	0	1.02	1.5	0.0242
1	5	1.05	5	0	1.02	1.5	0.0243
2	10	1.01	0.015	0	1.5	5	0.0246
1	1	1.1	0.015	0	1.02	1.02	0.0246

Appendix D Test of time intervals

Tests have been done on computational effort using intervals to store time-dependent travel times. Presented in this appendix is the time needed for generating and reading instances into the solver, as well as filesize for time-dependent travel time and memory needed for the solver. Values are presented both for approaches using all T and span of intervals. In addition, a comparison of time usage for 500 iterations in the solver are presented for approaches using all T and span of intervals.

Time is shown in milliseconds, filesize and memory in megabytes.

Table 22: Time intervals and corresponding test values for generating test instance, n = 58, density = 0.49

	Generate all intervals		**Generate span**	
Interval	Time	Filesize	Time	Filesize
1	9 552.7	27.092	7 336.2	1.688
2	6 701.5	13.616	5 881.5	1.688
5	5 204.6	5.530	5 132.0	1.688
10	5 006.9	2.837	4 681.2	1.688
15	4 549.3	1.940	4 576.6	1.676
30	4 538.3	1.041	4 254.7	0.859
60	4 339.1	0.587	4 147.0	0.415

Table 23: Time intervals and corresponding test values for reading instance into solver, n = 58, density = 0.49

	Read all intervals		**Read span**	
Interval	Read	Memory	Read	Memory
1	7 906.0	44.712	346.5	5.576
2	3 992.1	31.184	353.1	5.576
5	1 630.7	12.220	350.1	5.580
10	846.8	9.572	352.8	5.584
15	581.0	7.796	353.0	5.584
30	315.5	5.832	181.1	4.928
60	176.2	4.948	88.7	4.376

Table 24: Time intervals and corresponding test values for generating test instance, n = 116, density = 0.65

	Generate all intervals		**Generate span**	
Interval	Time	Filesize	Time	Filesize
1	110 942.8	243.989	48 693.7	8.515
2	69 275.5	122.555	34 338.6	8.515
5	40 583.4	49.698	31 041.5	8.500
10	31 698.4	25.410	25 676.9	8.492
15	28 197.2	17.336	25 168.4	8.467
30	25 097.8	9.215	23 939.8	3.907
60	24 349.6	5.156	23 857.6	0.415

Table 25: Time intervals and corresponding test values for reading instance into solver, n = 116, density = 0.65

	Read all intervals		**Read span**	
Interval	Read	Memory	Read	Memory
1	49 886.0	221.532	1 695.3	12.988
2	25 221.4	149.416	1 703.1	12.988
5	10 240.3	48.168	1 700.2	12.984
10	5 231.7	33.912	1 690.3	12.984
15	3 581.3	24.480	1 698.5	12.984
30	1 883.4	13.948	718.4	7.896
60	1 023.0	9.224	419.8	6.372

Table 26: Time intervals and corresponding test values for generating test instance, n = 273, density = 0.73

	Generate all intervals		**Generate span**	
Interval	Time	Filesize	Time	Filesize
1	899 920.5	1 561.308	339 367.7	42.485
2	635 423.1	784.568	270 521.0	42.481
5	302 315.7	318.287	202 975.9	42.441
10	238 974.7	162.853	186 430.0	42.388
15	233 681.1	110.992	181 233.9	42.313
30	230 623.5	59.121	168 919.9	25.416
60	214 251.4	33.245	162 817.1	17.169

Table 27: Time intervals and corresponding test values for reading instance into solver, n = 273, density = 0.73

	Read all intervals		**Read span**	
Interval	Read	Memory	Read	Memory
1	316 178.3	1 372.776	8 150.0	49.104
2	158 868.2	919.120	8 166.7	49.104
5	64 474.4	282.472	8 163.7	49.096
10	32 885.7	192.060	8 100.3	49.096
15	22 425.9	132.516	8 135.7	49.096
30	11 749.4	66.192	4 526.5	28.772
60	6 371.1	36.452	2 779.7	19.728

Table 28: Time usage for 500 iterations for Solver storing all T and using interval span

	All T	**Span**	**Increase**
OS-27	1531	1547	1.05%
OS-31	2094	2266	8.21%
OS-58	9051	9398	3.84%
OS-116	49020	52052	6.19%
Nortura-97	31343	35750	14.06%
Nortura-273	418654	434595	3.81%

Appendix E Test of computational effort

A regression analyzis have been done to find an approximate measure on how many iterations can be performed per second for instances of different sizes. Regression analyzis show that the number of iterations can be estimated as shown in Equation 20. Figure 19 show the number of iterations per second for the test cases generated in this thesis.

$$\#iterations = 1906252 * n^{-2.22} \tag{20}$$

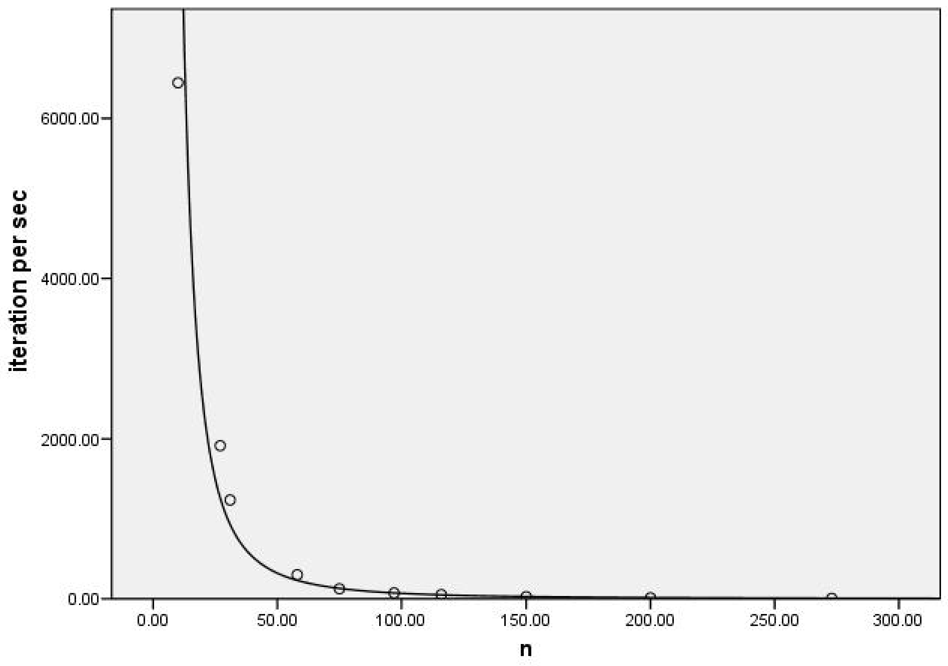


Figure 19: Prediction of iterations performed in one second.

Table 29 show how long the TDVRP solver run before stagnating, meaning the last best solution found. For all test cases, the solver is run for 1 hour. To get significant data for regression analysis, 4 random test-cases have been made of size $n = 10, 75, 150, 200$.

Table 29: Milliseconds until the TDVRP solver stagnate.

Instance	n	**Density**	**Time**
TestCase-10	10	0.60	0
OS-27	27	0.03	78
OS-31	31	0.43	1735
OS-58	58	0.49	66530
TestCase-75	75	0.75	27094
Nortura-97	97	0.74	458313
OS-116	116	0.65	97343
TestCase-150	150	0.72	81312
TestCase-200	200	0.69	556672
Nortura-273	273	0.73	1175860

Tests are run on Intel®Xeon®CPU E5450 3.0GHz with 8GB RAM, using MinGW GCC 5.4.1 compiler.