CHALMERS

A Complete Customer-to-Customer Shipment Scheduling Module Solving the CVRPTW and MSTPP problems

Master of Science Thesis

MATTIAS SJÖBLOM
MARCUS HEDENSTRÖM

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
Göteborg, Sweden, March 2012

A Complete Customer-to-Customer Shipment Scheduling Module Solving the CVRPTW and MSTPP problems

MATTIAS SJÖBLOM
MARCUS HEDENSTRÖM

Examiner: Peter Damaschke

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Department of Computer Science and Engineering
Göteborg, Sweden March 2012

Abstract

We implement a module for scheduling orders within a transportation network, considering both a pickup route between customers and a terminal, and the transportation between terminals. Our goal is to schedule the full set of orders, minimizing the total cost by having orders share transportation vehicles when possible, even in the non-trivial case when some orders need to be diverted from their own shortest path to do so. We evaluate the effectiveness of a genetic algorithm on the vehicle routing problem, and implement a serialization protocol.

Keywords: MSTPP, steepest descent/greedy algorithm, k shortest paths, vehicle routing, CVRPTW, Genetic algorithms,

Vocabulary

Carrier a service transporting shipments between two terminals

Location a geographical site of interest to the module

Cross-dock a location with multiple incident carriers, where shipments are exchanged

Schedule mapping from orders to routes

Depot a facility that provides storage and dispatches vehicles

Route a sequence of carriers assigned to an order

Trip a sequence of addresses visited without returning to the depot

Order an item to be sent from one location to another

Shipment a unit of one or more orders

Leg the smallest unit of movement, between two locations without intermediate states

VRP vehicle routing problem

CVRPTW capacitated vehicle routing problem with time windows

Terminal a location with only entering or leaving carriers

Contents

1 Introduction **1**

1.1 Purpose 1

1.2 Delimitations 3

1.3 Domain model 3

2 Method **5**

2.1 Last-mile routing 5

2.2 Terminal-to-terminal routing 6

2.3 Multi-stop 9

3 Detailed Method **11**

3.1 Using genetic algorithms 11

3.2 Solving the Modal-shift transportation planning problem 14

3.3 Finding the k shortest paths 16

3.4 The user interface 17

3.5 Communication 18

4 Results **19**

4.1 The Vehicle Routing Algorithm 19

4.2 Solomon test instances 20

4.3 Multi-stop 20

4.4 Service company 20

4.5 Terminal-to-terminal routing bounds 21

5 Discussion **22**

5.1 Setting up the genetic algorithm 22

5.2 Expandability 23

5.3 Cost structure 25

6 Conclusions **27**

1 Introduction

Scheduling orders through a network of carriers and locations, while consolidating orders into larger consignments to reduce the total cost, is a computationally hard problem. The routing of an order involves several steps. To start with, the shipment is picked up by a vehicle at the origin address and handed over to the terminal in charge of the area containing this address. From the terminal one or several shipments should be routed over a network of terminals and cross-dock locations. When a shipment arrives at its final terminal, it will be delivered to the destination address.

1.1 Purpose

In this thesis we will design and implement a software component for scheduling a set of orders, with the intent of minimizing overall business costs. These costs come from two different areas. First, there is presumably money to be saved by driving shorter distances, and reducing driving or idle times for each vehicle. These costs are the result of the planning done in advance by the shipping company, when the schedule was set, and not directly a consequence of how the calculations were made. Second, if a person is assigned the task of calculating the schedule, there is an additional cost per man-hour to be considered. Both of these costs are within the scope of this thesis.

The outcome of this thesis, a software component which integrates with existing tracking and planning systems, should address both goals, by using efficient algorithms to calculate the schedules, and saving man-hours by automating as much as possible of the process. The practical part of the thesis is done at Surikat AB, a software company located in Göteborg, Sweden. The existing system which the component will be integrated into is developed and maintained by Surikat AB.

Two rules of thumb, both intended to reduce overhead costs by aggregating orders together, will be used to define a set of subproblems. Constraints are placed on the problems to reflect the physical nature of the transportation domain.

At a local level, it is advantageous in terms of travel distance for a vehicle to pick up (or at the other end of the journey, deliver) as many orders as possible in a single tour before returning to the terminal. The length of each such tour is limited by the capacity of the vehicle, and should operate within a set of geographically clustered addresses. The pick-up and delivery of orders within such an area is transferable

to the well-known Vehicle routing problem (VRP) [8], which will be solved with a genetic algorithm, described in the method section.

There are however at least two things about the real-world application that may be overlooked in the simple form of VRP. For one thing, the vehicle cannot travel infinitely fast, and as the pickup destinations may want the order picked up before lunch, or after five in the afternoon. Thus, the vehicle will need to schedule visits in some order, such that every location is visited within its specified time window (TW), and the vehicle still has enough time to reach all of them in time. Then, even if the vehicle makes the visits on time, there is no guarantee the vehicle has room for every order, unless this was wisely calculated in advance. Anyways, this problem and our implementation of it will also consider vehicle capacity (C), and provide additional vehicles. These two factors extend the problem to Capacitated VRP with Time Windows, or CVRPTW.

At the global level, consisting of the cross-dock network, it is supposedly beneficial to schedule multiple orders at a time rather than processing them individually. That way, orders that share origins or destinations, or travel along common routes, can be merged into larger consignments, to better utilize the load capacity of the available carriers. This problem is known as the Modal-shift transportation planning problem (MSTPP) [6].

In this thesis, we will implement and analyze a system with algorithms solving the aforementioned problems, as well as provide a user interface for monitoring the finished schedule, with the goal of reducing the man-hours spent on manual scheduling. It is expected that this system will be fast enough to make modifications, such as adding a new order or changing the scheduling manually to adapt to external preferences, and allow the user to see the result of such modifications in near-real time.

1.2 Delimitations

The intention of the project is to work with fourth-party logistics operators (4PL) [10], meaning that we do not operate any vehicles or equipment ourselves. Rather, when scheduling shipments we simply purchase part of a vehicle or conveyance from 3PL or 2PL companies. The company selling transportation space will make their own analysis on how much to charge for each unit of vehicle space, in a process not transparent to us as buyers, and thus will not influence the decisions of our system.

Two approaches to solving this problem will be considered. Either the problem is solved for 3PL or 2PL agents, with the assumption that the obtained solution will also be a close approximation for the fourth party agent, or the algorithm could be adapted to accept cost data from an opaque module provided by the third party shipping company. In the latter case, the problem is still a valid MSTPP problem, only that costs are provided externally.

1.3 Domain model

As a key input data type, orders define start and destination addresses, as well as the weight of the physical item, and time windows within which the order is ready to start its journey and when it should be delivered to the recipient. We also refer to orders as shipments when they are in transit, potentially as part of larger consignments.

The network of cross-dock locations is represented by a graph, with a vertex for each cross-dock or terminal, and an edge for each carrier between two vertices. A carrier has information about the cost of the transportation and the time it takes to send a shipment from the start location to the end location. The load capacity of the transportation conveyance is also specified. Between two locations there can exist several carriers, which represent different companies that offer transportation between the two locations.

Locations can be divided into three different types, which we use in this thesis, terminals, cross-docks and addresses. Terminals and cross-docks form a network of locations where cross-docks are the core of the network. Terminals, the leaves in the cross-dock graph, are connected via carriers to one or several cross-docks. If shipments share the same path out from a cross-dock consolidation of those shipments is possible. Naturally also de-consolidation can be done at cross-docks or terminals. A cross-dock can also represent several special case locations such

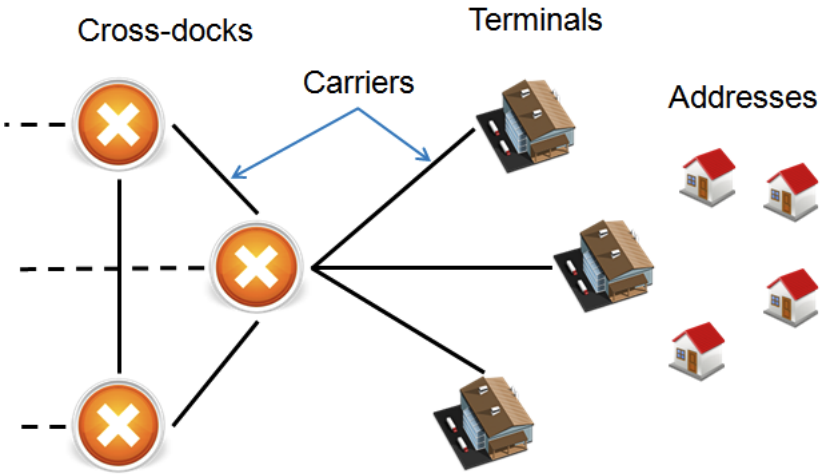


Figure 1: Basic domain model.

as border customs, which shipments have to pass through to travel between two countries.

Addresses are often customers or other forms of locations, which represent source and destination addresses of orders. Each address is within reach of a terminal, which is in charge of an area of addresses that are clustered geographically. Carriers between terminals and cross-docks are rather static; costs and other information depend on what companies offer transportation of shipments between the two locations. Transportation information between terminals and addresses are based on travel distance, speed limits and often a limited number of available vehicles.

The solution will contain trips, legs and routes. At the terminal a vehicle or carrier performs a trip when picking up or delivering orders and in the end returns to the terminal. Legs represent a used carrier together with all shipments to be sent with a carrier. A routes consists of several legs, which can for example represent a full path for an order.

2 Method

The routing of shipments was divided into two problems, one concerning the pickup of orders and gathering them into terminals, as well as delivering orders from the destination terminals to the customers, and the main problem of scheduling vehicles through the cross-dock network. The different problems are described in this section.

Orders were transported between addresses and terminals by one vehicle or a fleet of vehicles, in a part of the operation called last-mile routing. Then shipments were routed through the cross-dock network described by the terminal-to-terminal routing.

Some orders were delivered to and from terminals by an outside service, which was not within the knowledge of the system. These orders had source and destination addresses set to a terminal direct. In other words they were inserted into the terminal to terminal routing without consideration of last-mile routing.

As an alternative to the routing process of combining last-mile routing with terminal-to-terminal routing, multi-stop routing was introduced. Multi-stop routing mainly concerns larger shipments, which would not gain any cost reductions by traveling through the cross-docks network. For example shipments that fill or almost fill a vehicle, with clustered sources and clustered destinations.

2.1 Last-mile routing

The pick-up and delivery at source and destination terminals are handled separately from the rest of the routing procedure, so the final destination of the order is not important. The only objective is to collect every single order from the region near each terminal as efficiently as possible. The variables that make each solution unique are the total number of vehicles used, the subsets of customers serviced by each vehicle, and the relative order of visits to those customers by the vehicle. After finishing the local pickup rounds, the system will proceed to schedule transportation between terminals.

The problem of routing a vehicle between addresses within an area is known as the Vehicle Routing Problem (VRP). Temporal constraints for when the vehicle must visit each address to be able to pick up or deliver orders in time are introduced. Also, a constraint for maximum load of a vehicle is taken into account. An instance

of VRP with these constraints is called Capacitated Vehicle Routing Problem with Time Windows (CVRPTW) [2]. Orders start their journeys at an address from which it needs to be shipped to a nearby terminal. In the end of a shipment's route, at its destination terminal, the shipment needs to be split into individual orders and delivered to each destination address. The pick-up and delivery can be modeled as the same problem, one or several vehicles will travel from the corresponding terminal to all addresses to fetch or deliver orders and finally return to the terminal.

In an area of n customers, there are n! possible permutations of customers (trips) for a single vehicle. However, when any number of vehicles taking one trip each is allowed, for each such permutation there will be multiple partitions into trips, increasing the number of possible solutions. Each customer in each one of the permutations is free to choose any vehicle, thus multiplying the total number of solutions by n^k, for k vehicles.

On the other hand, the solution does not care how the set of trips is ordered, i.e. vehicle 1 servicing customers 1, 2 and 3 while vehicle 2 services customers 4, 5 and 6 is indistinguishable from the similar solution where vehicle 1 services customers 4, 5 and 6, and vehicle 2 services customers 1, 2 and 3. Therefore, the number of possible solutions is reduced by the number of possible orderings of vehicles by removing some of the partitions of each original permutation. Because of the very large number of possible solutions, for even a moderately small set of customers, it would be too costly to exhaustively evaluate every one of them. Instead, a genetic algorithm is used to evolve a solution [5].

Addresses were taken as genes, comprising chromosomes which in turn represented the route in order of ascending pickup time. The genetic algorithm then evolved a population of routes, each evaluated for its fitness according to the total travel time, with additional time added as a penalty whenever the vehicle arrived late at a location.

2.2 Terminal-to-terminal routing

The cost per used volume or weight of a carrier, within the cross-docks network, is assumed to be cheaper the more shipments that are placed on the on the carrier. Hence a low load ratio of a carrier is more expensive per weight or volume unit than a carrier with a high load ratio. Therefore it is beneficial for shipments to share carriers if possible. We assume that in some cases shipments can use, individually seen, a more expensive route to be able to share a carrier with other shipments and still decrease the overall cost.

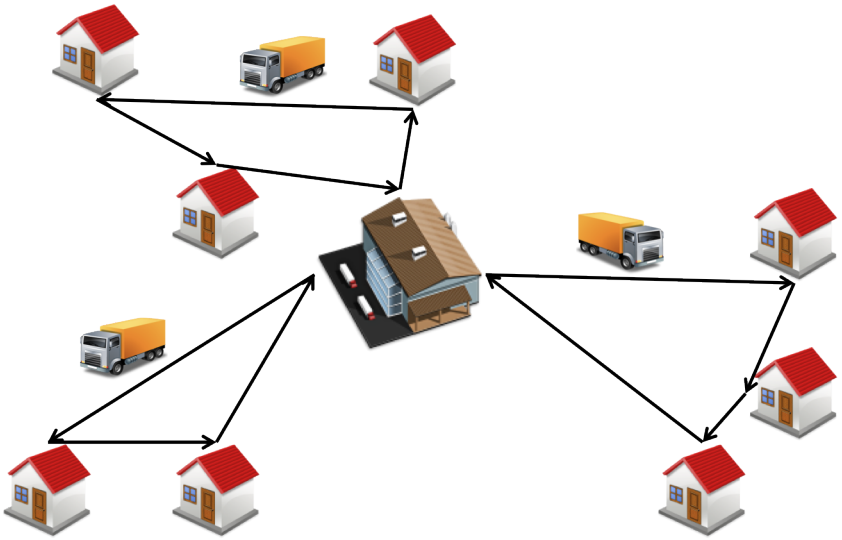


Figure 2: Vehicle routing problem. A solution to the vehicle routing problem is a set of trips, such that every customer is visited exactly once. The trips must meet time window requirements at each customer, and vehicles can only carry a certain amount of cargo. These constraints may make it necessary to use multiple trucks for the solution.

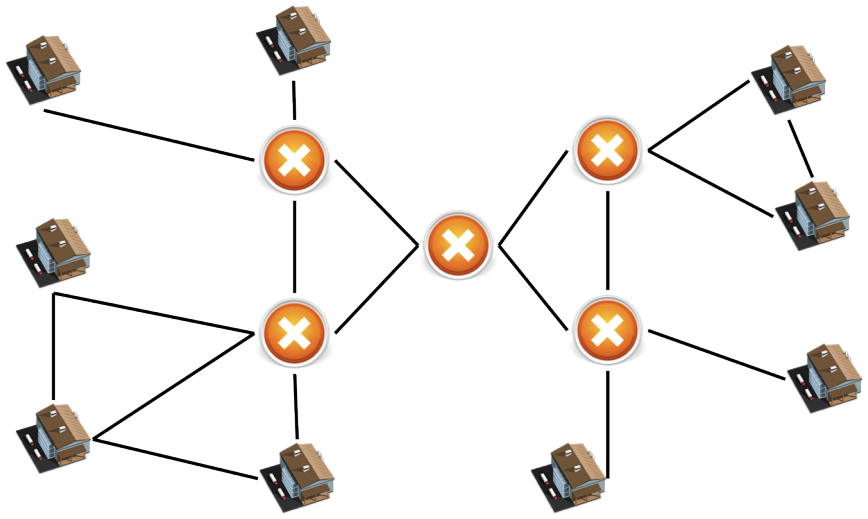


Figure 3: Cross-dock network. Every customer is services by one terminal, which may also act as a cross dock. Orders are collected by a set of vehicles into their designated terminal, and are routed through the cross dock network toward the terminal servicing the recipient, being exchanged by different carriers along the way, following the most efficient route.

When shipments finally were situated at corresponding starting terminal the routing in the cross-dock network were run. Since the price per used space of carriers decreases the higher the load ratio was, a natural goal of the algorithm was to consolidate shipments as much as possible. On the other hand, shipments cannot travel around the network just to fill vehicles without getting the total price of the schedule to suffer. Hence a combination of finding as cheap route as possible and consolidate shipments as much as possible without increasing the overall cost were the final goal of the algorithm.

For each order, consider a list of all possible routes to its destination terminal, sorted in increasing order of a cost parameter. The order of these candidate routes is guaranteed by the k shortest paths algorithm, such that the i^{th} candidate is at least as good as the $i+1^{th}$ candidate, where "good" means lower value of whatever parameter it is the goal to minimize, typically the distance of a route.

A number of routes were selected from the top of the list, as candidates for each shipment. For example 5 cheapest routes and 5 fastest routes could be chosen. Also predefined routes can be inserted as a complement to the selected routes or strictly wanted routes would be inserted as only candidate routes. Each route consists of one ore more legs separated by cross-dock locations, where consignments can

be transferred from one vehicle to another. In turn, each leg has a set of carriers assigned to it, and for each order any one of those carriers can be selected for moving the order along.

As described by Amano et.al. [6], the solution space for this problem grows very fast as the number of orders increases, and when routes are made up of a larger number of legs. The same authors also recognize that the MSTPP is in the NP-hard class of problems, since it includes the bin packing problem. To improve the performance of the algorithm, they propose a heuristic algorithm - the steepest descent algorithm - for selecting candidate routes, and a greedy algorithm for assigning orders to carriers. After scheduling orders to carriers for each leg the steepest descent algorithm is run again to see if the credentials to choose candidate routes has changed. If new candidate routes are chosen the greedy algorithm is run again. This loop continues until the same candidate routes are chosen twice in a row. When done the chosen carriers for each leg for each order is presented as a solution.

2.3 Multi-stop

An algorithm that solves the multi-stop problem was implemented. The multi-stop problem is a simpler situation than the earlier described last-mile routing combined with terminal-to-terminal routing. Instead of routing shipments through the cross-docks network a vehicle is routed between a few locations where shipments are loaded onto the vehicle until the capacity of the vehicle is reached. Often the vehicle travels to another area where it delivers the shipments to their respective destination addresses. Also a backhaul situation appears when the vehicle travels back to its starting area, hence the algorithm can be run for shipments that has the opposite source and destination areas to the initial shipments which was newly delivered.

The multi-stop problem is similar to the vehicle routing problem, thus parts from the genetic algorithm implemented for CVRPTW were reused. After collecting shipments at sending customers the vehicle will, instead of returning to a depot as in the VRP, continue to the next area of customers where shipments will be handed out to the correct receiving customer. It is assumed that multi-stop scheduling often concerns larger consignments. Therefore it is important to fill the vehicle in the opposite order as it will be emptied to avoid unnecessary reloading of consignments at stops. This is taken into consideration when finding a solution. The cost parameters concerning travel from source of order A to source of order B is

therefore influenced by the cost parameters by traveling from destination of order B to destination of order A.

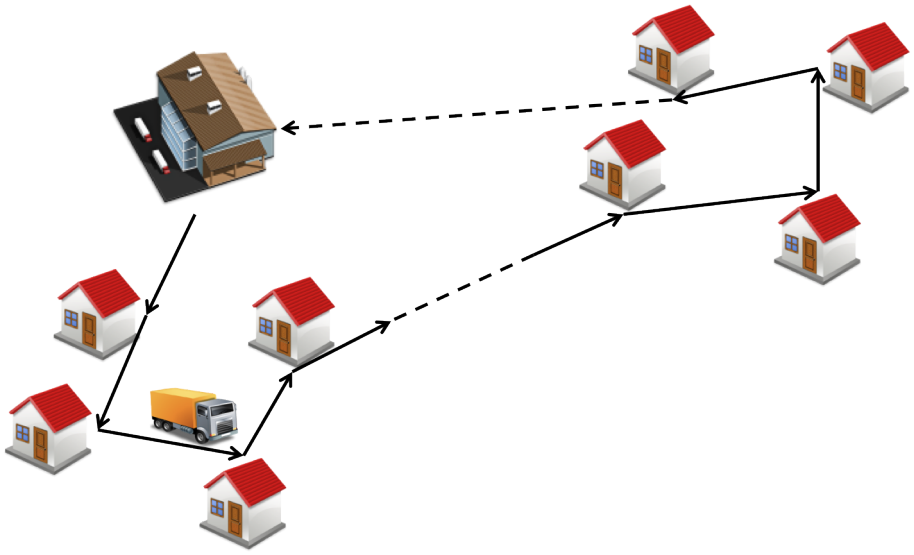


Figure 4: The multi-stop problem. Customers are partitioned into two sets, one containing all senders and another containing all recipients. All orders are picked up from customers in the first set by a single vehicle, before any deliveries are made. Once finished, the vehicle travers to the second area and delivers the orders in reverse order. Finally, the vehicle returns to the terminal.

3 Detailed Method

In this section algorithms and solution implementations previously described in the method section are reiterated with a higher level of detail. Also GUI, and communication protocol implementation are explained here. Readers not interested in the details of the implementation could skip to the next section.

3.1 Using genetic algorithms

For the genetic algorithm, chromosomes were defined as a sequence of addresses. One address, the one representing the single depot, was the only gene allowed to appear multiple times in the same chromosome, while all other addresses appeared exactly once. Following the addresses of a chromosome in order defined the route, visiting every customer exactly once, with intermittent visits to the depot.

When evaluating a chromosome in for example the CVRPTW problem, its fitness value was the total travel distance and the total time spent servicing all customers, with the strict requirement that customers be served within the established time windows. If the vehicle arrives early, it must wait until the customer is ready before servicing it, and thus the departure time from that customer is also postponed by an equal amount of time. On the other hand, if the vehicle arrives late, it is penalized by adding a number to the total fitness value. In other use cases the fitness value of the genetic algorithm can be described in another way depending on what should cost variable that should be optimized.

As opposed to many other problems that can be solved with genetic algorithms, where chromosomes are composed of genes that may take on arbitrary values, the vehicle routing problem requires that each address be visited exactly once. For instance, if two chromosomes were used as parents to produce two offspring through single-point crossover, in other words combining the first part of one parent with the last part of the other, the offspring might inherit some genes twice, and some not at all.

Therefore, the genetic algorithm replaced simple crossover operators with custom operators that preserve the individual locations, such as the sequence based crossover (SBX) operator [4]. This operator selects two points, one on each chromosome involved in the crossover, at random. Each point will separate a trip into two parts, one going out from the depot and one returning to the depot. First, the outgoing part of the first trip is connected to the returning part of the second, form-

p_1	a	b c d	e f g h
p_2	b	f a g	c h d e
	-	b c d	- - - -

(a)

p_1	a	b c d	e f g h
p_2	b	f a g	c h d e
	f	b c d	- - - -

(b)

p_1	a	b c d	e f g h
p_2	b	f a g	c h d e
	f	b c d	a - - -

(c)

p_1	a	b c d	e f g h
p_2	b	f a g	c h d e
	f	b c d	a - g -

(d)

p_1	a	b c d	e f g h
p_2	b	f a g	c h d e
	f	b c d	a h g e

(e)

Figure 5: PMX crossover example

ing a hybrid trip, removing duplicate genes in the process. Second, duplicates are eliminated from the rest of the chromosome, favoring the instances inside the new trip. Finally, addresses that were disconnected from their trips by the crossover, and no longer present in the child, are inserted by best-fit selection, where the detour incurred by adding the address is minimized.

Another crossover operator Partially Mapped Crossover (PMX) [9] was applied to the system. The process of the operator can be explained by two simple examples.

Given a pair of individuals (P_1 and P_2) from the current population, a randomly selected range of genes is copied from P_1 to the new chromosome (Figure 5a). For each position p in the range, the gene occupying position p in P_2 is copied to position p in the new chromosome (Figures 5b, 5c, 5d). The remaining genes are copied directly from P_2 (Figure 5e), and the new chromosome is complete.

A special case occurs when same allele in the selected range of P_1 is also in the the range in P_2. For instance, allele c is found inside the selected range in both P_1 and P_2. In this case c is temporarily ignored, but the other genes are copied (Figure 6a) resulting in duplication of those alleles in the child (Figure 6b). As in the last example, the remaining genes are copied form P_2 (Figure 6c). Now there are duplicates of the c allele, and the a allele is missing altogether. The c allele outside the initial chosen sequence is replaced by the a allele (Figure 6d).

In addition to recombination, a simple swapping mutation operator was applied

p_1	a b	c d e	f g h
p_2	b f	a g c	h d e
	- -	c d e	- - -

(a)

p_1	a b	c d e	f g h
p_2	b f	a g c	h d e
	- -	c d e	- g c

(b)

p_1	a b	c d e	f g h
p_2	b f	a g c	h d e
	b f	c d e	h g c

(c)

p_1	a b	c d e	f g h
p_2	b f	a g c	h d e
	b f	c d e	h g a

(d)

Figure 6: PMX crossover example (special case)

to some chromosomes in each generation, to encourage exchange of addresses between trips. This operator simply chooses two locations at random, and swaps the corresponding genes.

To help finding better solutions an inversion operator was added. It is helpful when a chromosome contains a desirable sub path, but in the wrong direction, for example such that it gets increasingly harder to make deliveries within the time windows. The inversion operator takes a random range within the chromosome and reverses the order of the genes within that range, making it possible for any backwards paths to be corrected, and better contribute to the chromosome's fitness.

Initially, two chromosomes comprised the population; one where the sequence of addresses was broken up into trips by equally spaced instances of the depot allele, and one where the genes were inserted in a best-fit manner, using the push-forward insertion heuristic (PFIH) as proposed by Solomon [8]. This more informed method starts with an empty trip with no locations, except for an instance of the depot at each end. Then, while there are still unrouted addresses left, the best location for one such address in the trip is determined, such that the sum of the extra distance traveled and the delay incurred by the detour is minimized. To avoid preempting the insertion of addresses that may fit that slot better, such an address, if it exists, is selected instead of the prior address selected for insertion into the current trip. In the case that no address is compatible with the current trip, meaning that whichever address is inserted at the optimal location, the trip cannot service all addresses within the time windows, the current trip is added to the set of finalized trips, and a new empty trip is started. The benefits of this method come from the fact that the number of trips (or vehicles used) is kept to a minimum, and every address is serviced within its time window. However, further computation using a genetic algorithm is needed for improving the chromosome to

a less costly route. Note that only the latter method can reliably produce feasible solutions, while the former makes no effort to ensure service within the given time windows for each address.

Finally, the population was allowed to evolve until it reached a plateau, where each successive generation did not produce a better individual than last over a set number of generations, or until a predetermined number of generations in total had passed. The fittest chromosome was then provided the final solution to the problem.

3.2 Solving the Modal-shift transportation planning problem

The goal of the MSTPP is to schedule a set of shipments to a set of carriers, such that each shipment arrives on time, and the total cost is minimized. For this sub problem of the project, shipments are assumed to originate at a source area, and are to be transported along a route to its destination, which is a shipping terminal near the end customer. Each route can have an arbitrary number of legs, and carriers are selected per leg. This means that any pair of legs is joined at a cross-docking facility, where shipments can be exchanged between carriers.

The algorithm proposed by Amano et.al. [6] selects a small number of candidate routes for each order. These candidates include the shortest travel time route and the least expensive route, both of which can be derived from a weighted graph using Dijkstra's algorithm. However, more than one of each kind is needed, so instead of Dijkstra's algorithm, an algorithm to calculate the k shortest paths [3] (described later) for some chosen number k was used to populate the set of candidate routes. A steepest descent algorithm is used for finding a solution. Each shipment is given a route, which together with all other shipments minimizes the overall cost of transportation.

For each shipment all candidate routes are evaluated and given an initial cost, which they are sorted by. The cost is the lower bound cost of the route, i.e. the cost per weight unit is calculated as if each carrier would be fully loaded including the shipment (see Equation 1) and the cheapest carrier for each hop in the route is used. Also a time hardness of the route is added to the total cost. The time hardness is calculated as the difference between a shipment's earliest possible starting time and the latest time it has to start from its starting location to meet the deadline at its destination. The time hardness is only used when calculating the fictive cost of a route, when calculating the cost of a single carrier it is not taken into account.

$$c_w = \frac{cost\ of\ the\ full\ carrier \times weight\ of\ shipment}{capacity} \tag{1}$$

As an initial setting each order is paired with its cheapest candidate route. This setting is scheduled in a greedy algorithm where suitable carriers are chosen for the transportation of all shipments.

The cost of each chosen candidate route is updated accordingly to the greedy algorithms selection of carriers and the scheduled load of carriers. If any of the candidate routes is more expensive than before, the candidate routes are resorted and a pair of each shipment and its cheapest route is chosen again as input to the greedy algorithm. After the first time the greedy algorithm is run most candidate routes are likely to get more expensive since the lower bound cost was used when initialized.

When no candidate route gets more expensive after the greedy algorithm has selected carriers the steepest descent algorithm is done scheduling. The solution is generated from the set of chosen carriers that will transport the shipments; shipments can and most likely will share carriers. Since it is cheaper to transport many shipments than few on one carrier the algorithm is likely to choose carriers which can be used by several shipments.

The greedy algorithm iterates over a list of shipments in ascending order of earliest starting time at current stop. A shipment is placed on a new unused carrier or an already used carrier with other shipments on, the carrier with the cheapest cost per unit weight is chosen. The chosen carrier is set ready to load more shipments.

When there are no orders possible to place on a carrier C, due to time restrictions or load restrictions, a better carrier, B, is searched for. The better carrier must fit all shipments that are scheduled on C and must agree with all time restrictions of the shipments and finally it must be cheaper per load unit than C. If a better carrier is found, all shipments on C are reloaded on B, which is set ready for loading more shipments. If no better carrier is found then C is sent for departure and each shipment travels one stop on its candidate route. C is a part of the solution schedule of the greedy algorithm. When a shipment travels one stop it is placed back in the ordered list of all other shipments to be further scheduled through its route. If a shipment arrives at its destination it is a part of the solution schedule and not put back into the list of orders.

3.3 Finding the k shortest paths

Finding the shortest path between a source and a target vertex can be achieved by Dijkstra's algorithm. However, it is not trivial to derive the next shortest path, the next one after that and so on. A different algorithm proposed by Eppstein [3] computes the k shortest paths in $O(m + n \log n + k)$ time.

The algorithm starts at the target vertex, tracing out the shortest paths from every other vertex in the graph using a simple shortest path algorithm, such as Dijkstra's algorithm. These paths form a tree containing the same vertices as the graph, and a subset of its edges, called the shortest path tree. The target vertex is the root of the shortest path tree. Thus, from any node it the graph, it is possible to repeatedly move to the vertex's parent in the shortest path tree, eventually reaching the target after visiting a unique sequence of nodes.

The edges in the graph that are not part of the shortest paths tree are considered sidetracks, with the common trait that by moving along them on route to the target will result in a total cost that is greater than the shortest path cost from the same starting vertex. The amount of cost that is added is denoted by δ, and can be computed for every path in the graph. The δ parameter for the edges that are in the shortest path tree are zero.

The algorithm starts with the shortest path from source to target, that is, the path that have no sidetracks. For each sidetrack that is incident to any vertex on this path, including the sidetrack, and otherwise following only paths in the shortest path tree form an alternative path. These paths are arranged in a tree structure, the path tree, with the shortest path at its root and the other paths as children of the root path. Given any path in the path tree, a new path can be formed by adding an additional sidetrack.

Next, each of the paths defined by its specific sidetrack is used to form new paths with an additional sidetrack, by adding one child path for each possible sidetrack that occurs on the part of the parent path once its sidetrack has been passed. That way, there is no risk of the same sidetrack being used more than once. This process continues for each of the children until there are no sidetracks left on the remainder of the paths to the target node.

The path tree now contains all possible paths through the graph, from the source to the target node. Furthermore, the children of a node has one additional sidetrack in addition to the sidetracks inherited from its parent, and is therefore at least as costly (and in reality almost certainly more costly). Thus, the tree is actually

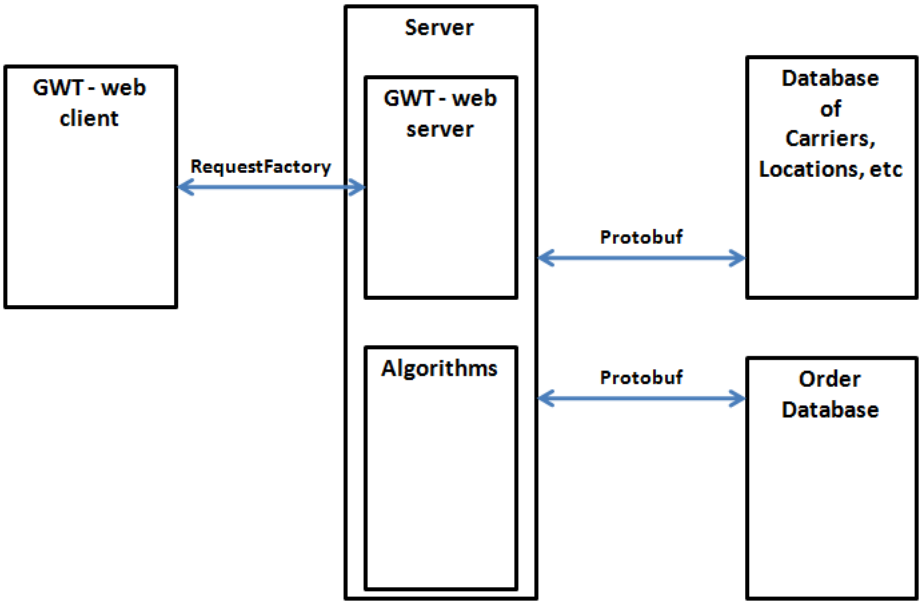
a heap, although an incomplete one of undefined degree. Conversely, if it were a complete heap of some defined degree, it would suffice to perform breadth first search until a desired number k of paths were visited, obtaining the k shortest paths. However, since the heap is irregular, more operations are required. By forming a single multigraph, such that a sidetrack edge reachable in the original graph is also reachable in the multigraph. Finally, performing a breadth first search in the multigraph, noting all paths explored, produces a list of the shortest paths in the original graph.

3.4 The user interface

The user interface was developed within the Google Web Tools framework (GWT) [1]. Solutions to each problem are fetched from the server, and presented visually on a map. All computation is done at the server, and the client is only allowed to access the finished results. Orders are not entered through this web interface, but are accessed from a database directly by the server.

Some solutions consist of separable units, such as the solution to the vehicle routing problem, where multiple vehicles may be needed to serve all addresses, and the trip each vehicle makes can be presented independently from the other trips. The user interface presents each unit of a solution, with the option to display either all at once, or just a specific unit.

3.5 Communication



The project was implemented in Java, separate from the user interface for modularity. Communication between the web client and the server was done with GWT's RequestFactory framework, with the benefit of easy representation of the domain model by client-side proxy objects. Requests are made by selecting a problem type, as well as problem parameters from the web interface, and waiting for the server to respond.

The environment of a scheduling problem was provided to the system in a serialized form. The system got carrier and location data from an external system and orders were also received from a third system. The protocol was described in Protobuf and implemented with help of Protostuff [11]. Protobuf is a bandwidth-efficient communication protocol, which can be used when serializing data to a file or sending it directly over a network.

Data was received from two external servers. One server, which provides the world of locations and carriers; this information was intended not to be updated often since the world of areas and available lanes should be rather static. Another server provided the orders, which were to be routed in the world. In the testing phase these two servers were simulated by one physical server and the same software. The system was made without any dependencies of the servers providing data; hence it should be possible to use any data source, as long as it is introduced to the protobuf protocol.

4 Results

The two sub algorithms we created was tested with different test data. The Vehicle Routing Algorithm was tested with three different types of data sources while the Terminal-To-Terminal Routing algorithm was tested with manually created sample worlds.

4.1 The Vehicle Routing Algorithm

The first out of three types of problems which we tested the algorithm with was the CVRPTW, using the Solomon instances [7] as input. These problems concern worlds with a single depot and typically a hundred customers to be served by an arbitrary number of vehicles. The goal is to minimize the total distance traveled, with the constraints of customer time windows and vehicle capacity, which affects the final number of vehicles required.

The second type was the so-called multi-stop problem, involving a single vehicle and two disjoint sets of locations. The vehicle was to pick up a shipment at each one of the locations in the first set, and deliver them to corresponding locations in the second set. In this problem, the vehicle acted as a last-in-first-out queue, meaning that the complete ordering of locations was defined by the first half of the route alone, with the second half visiting the corresponding locations in reverse order. Of course, the fitness of a solution still took distances traveled within the second set into account.

Finally, the algorithm was tested with real-world data sampled from a company operating service vehicles within a couple of different cities. These instances imposed no time windows on when customers were to be served, but simply guaranteeing one occasion at a weekly basis. Still one time constraint was applied; a trip was limited to a full work day, which in our case was 8 hours. As with the first type of problem, it was again possible to travel directly between any pair of locations at real-world distance and travel time.

4.2 Solomon test instances

We performed benchmark tests of the algorithm used to solve the vehicle routing problem, or the problem of distributing orders with a few vehicles within a local area, and compared our results to known solutions to the instances defined by Solomon [7]. These instances feature a random distribution of addresses, uniformly distributed or clustered, with a time window for each address. Any solution to one of these instances must visit every address, and must do so within the defined time windows. Any solution, or perhaps non-solution, that fail to arrive at some address within that address' time interval cannot be considered a candidate for a benchmark.

The Solomon instances come in different sizes, originally with one hundred address and one depot, but also reduced instances with only twenty-five addresses and one depot. While developing and testing the genetic algorithm, we also created a very small instance of seven addresses and one depot, but otherwise similar to the Solomon instances.

4.3 Multi-stop

Visually it was hard to determine how the algorithm performed on the multi-stop test cases since the order of addresses were influenced by both the pick up and the delivery area. Hence it is difficult to do a manually route vehicles in an as efficient way as possible. Our algorithm managed to produce a solution which according to the total distance traveled managed to shorten the distance significantly compared to manually routing of the vehicle.

4.4 Service company

Here the routing algorithm performed as expected, producing a solution that was about 17 percent more profitable than the manually constructed schedule. Worth noting is that the manually constructed schedule has been thoroughly refined over several years.

A special situation appears when a customer requests more than one service occasion per week. The genetic algorithm would sometimes place the two instances of this customer's gene right next to one another, since it saves travel distance to service it twice without servicing any other customer in between. But surely

the customer wants the separate service occasions at least on different days, for example one on Tuesday and one on Friday. To resolve this conflict, whenever the same customer appears more than once, one of the duplicate instances is removed from the individual before the algorithm is run. Later it is reinserted where it incurs the least extra cost, choosing from trips that do not already contain it. This is a greedy process, which lets the first duplicate customers choose their best possible position first, leaving following duplicates with whatever options are left. This step is performed until no trip contains duplicate instances of the same customer.

4.5 Terminal-to-terminal routing bounds

We assumed that the solution to the scheduling problem would lie between two extremes. A lower bound where every order traveled the shortest possible route in completely filled vehicles, thus minimizing the cost; a higher bound where each order still traveled the shortest possible route, but this time in vehicles carrying only the orders that would travel that route if they were scheduled on their own. The actual solution would cut costs compared to the higher bound by choosing beneficial routes for the total cost by achieving higher load ratios for chosen carriers. Still we did not expect all orders to be able to travel their individually cheapest routes or fill all used vehicles total capacity.

When running a sample world, comprised of 16 terminals and scheduling 1 000 orders, the lower bound cost proved to be around 92 % of our schedule cost after running the algorithm while the upper bound turned out around 112 %. This means that some orders were scheduled on not individually cheapest routes and consolidated together with other orders, reducing the total cost by about 10 percent.

5 Discussion

5.1 Setting up the genetic algorithm

The genetic algorithm used to solve the vehicle routing problem was given only four operators; the sequence-based crossover operator for forming new trips, a swapping mutation operator for facilitating moving locations between existing tours, an inversion mutation operator to inverse the order of parts of tours, Partially Mapped crossover operator which inheritances sequences from the parent individuals. The sequence-based crossover operator was the one that influenced the results the most.

A population of sample individuals was created, at first by simply listing the locations in undetermined order, with a set number of returns to the depot during the route, symbolized by additional instances of the depot in the chromosome. The primary drawback of this scheme was that the sample individuals were most likely infeasible, having the vehicle arriving at a location based on which locations were scheduled prior to it, and not taking the target time windows into account. Evolution of the population would, given the right fitness function, favor individuals with greater portions of their trips arriving on time, while other individuals, being penalized for missing time windows, would be eliminated from the population.

In testing the algorithm with Solomon benchmarks [8], no solution managed to reduce the penalty of the fittest individual to a negligible amount. In other words, the vehicle routing problems made by Solomon was not solved.

Another type of sample individuals was formed by the push-forward insertion heuristic (PFIH) [8], where they were composed from the bottom up, always making sure the current trip remained feasible, and choosing locations with best fit to be inserted into the trip. When a trip could not be extended, and remain feasible, the heuristic started a new trip.

This provided the population with individuals that were feasible from the start, and could only be improved on from there. However, the heuristic did not account for vehicle capacity, it principle assuming infinitely large vehicles. It was not investigated whether the heuristic could be extended to place limits on the total capacity of each trip, and start a new trip whenever the current trip reached the maximum vehicle capacity. Though in results with remaining penalty the capacity penalty is very small which proves that the PFIH together with chosen operators works well to minimize the capacity penalty.

To improve the solutions found by the algorithm one can add more mutation and crossover operators. With more operators a wider range of possible solutions is investigated. When handling a problem as VRP with time windows it has been noticed that operators with heuristics tends to perform better. Several operators are suggested by Potvin [8] and Starkweather [9]. Also different initial chromosome creators could be investigated to widen the search space from the starting population.

5.2 Expandability

While the basic algorithms are not meant to be modified, they are flexible in how different types of problems can affect their own control flow. For instance, the regular vehicle routing problem is different from the multi-stop problem with different ways of computing an individual's fitness. The first represents an individual as an ordered sequence of locations, which fitness is the total distance traveled, while the second problem has locations divided into two groups, representing an individual as a sequence of locations from only one of the groups. The fitness is computed as the sum of the distances traveled between these locations, plus the distance traveled between locations in the other regions, whose order is defined by the individual, though not explicitly encoded in its genome.

These different problems can be implemented with their own fitness functions, and even their own, custom genetic operators, and used by the common implementation running the genetic algorithm. This pattern allows the addition of similar problems, which may differ in how they compute fitness, and what genetic operators they use.

The method for populating the set of candidate routes selects routes in increasing order of negative impact they have on the value of the schedule as a whole. A natural parameter for ordering routes is the cost of traveling between the origin and the destination node along a certain route, where a less expensive route is selected over a more expensive route, thus keeping the total price of the solution reasonably low.

However, if one only selects inexpensive routes, the set of candidate routes may be biased toward routes that are slow, but economically attractive. For instance, the top routes could consist of cargo ships, which cost less per consignment, but take longer to reach their destination. To increase the diversity of shipping modes, only a subset of the candidate routes should be selected according to cost, and other subsets could be selected according to other desirable traits of a good candidate route, such as total transit time.

Within this module, the world is represented by a multigraph with locations as vertices, and carriers as edges. The weight of an edge is the value of the trait of interest when selecting desirable routes; cost, time, distance, number of nation borders crossed, or any other property that can be translated into a number. For each trait, a graph is constructed and used to find some number of paths, which add to the total set of candidate routes for the problem. The scheduling algorithm is then free to use whichever mode of transportation is more advantageous.

Though the module as tested only considers the cost of each carrier, and routes with other advantages were left out due to the input data providing only carriers of the same kind, adding routes with different advantages in recommended for future development.

5.3 Cost structure

As mentioned in 1.2 there were two approaches to handling the cost structure when routing shipments in the cross-docks network. Since the size of the test data provided was small it is hard to draw any conclusions of how well the algorithm operates in a different cost model than initially intended for.

Two approaches were investigated to solve the reconstruction of the MSTPP algorithm. First, by simulating a carrier's different cost steps by several different carriers the problem is narrowed down to the initial problem of paying for a whole carrier. The only different restriction is that each carrier have a minimal loading capacity. The disadvantage that of this approach is that the amount of carriers to choose from between two given locations increases dramatically. In a small world the algorithm does still run in acceptable running time but in a more realistic world the running time will increase fast.

The second approach and the one that is used in the final algorithm is to keep the cost structure of each carrier. When starting to load a carrier the cost is given by the cost of loading the initial shipments on the carrier. Later when adding more shipments to this carrier the cost will remain until enough shipments are loaded to reach the next step of the carriers cost structure. If there is any different carrier that is cheaper for all shipments loaded on the carrier it will be looked up on before dispatching the currently loaded carrier. If any cheaper carrier is found the shipments are reloaded to the new carrier. Hence a cheaper carrier will never be missed even though the costs varying by the load of the carrier.

It is worth noting that the problem of defining the cost structure in the last-mile routing was not a challenge. Since the vehicle is routed through a full trip the whole car is payed for. The sorting of orders or what size or weight the orders does not influence the cost. Therefore it is no difference of using a full or half-full vehicle since the vehicle is hired for the full trip.

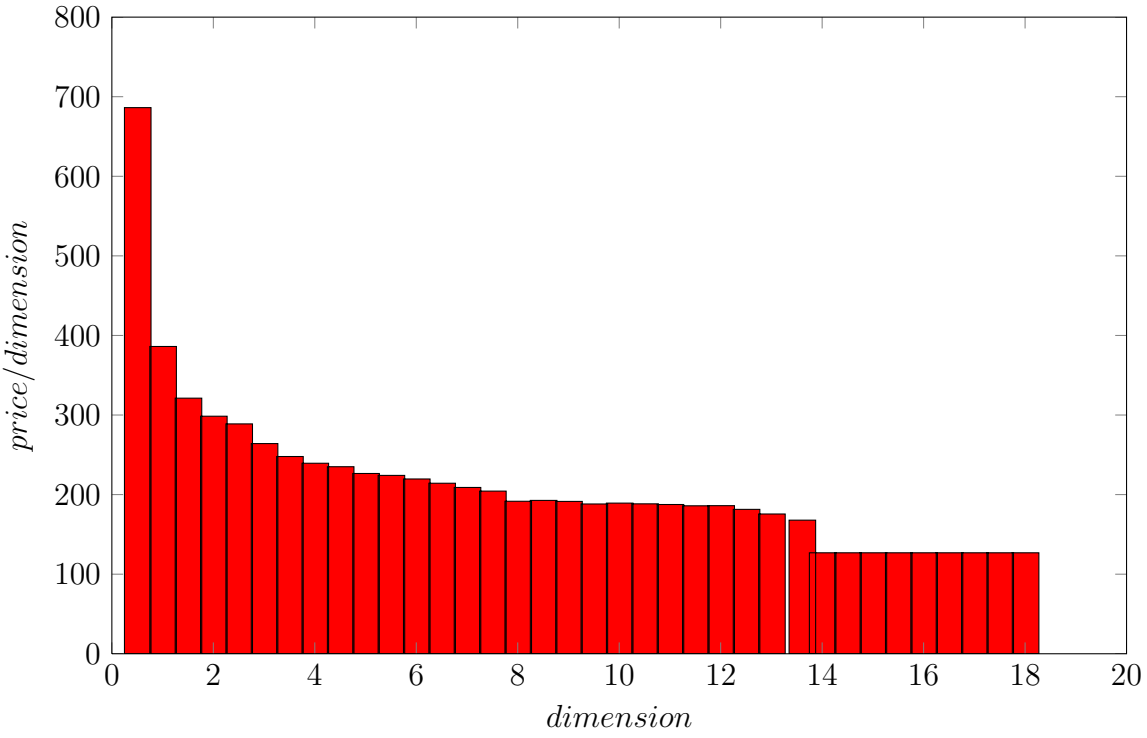


Figure 7: Cost structure. The price per dimension unit of the consignment, as a function of the total space purchased.

6 Conclusions

A genetic algorithm solves the different types of routing problems effectively, when given the appropriate genetic operators to produce variations of the population at each new generation. It is particularly successful when an initial population is constructed using some heuristic to generate feasible solutions, instead of starting with completely random individuals. For all type of vehicle routing problems, we are limited to genetic operators that preserve the set of alleles (the values of the genes), changing only the order within the chromosomes, and not the values of the genes themselves.

We also discovered that some types of genetic operators, especially the Partially mapped crossover operator, are harmful on individuals containing multiple instances of the same gene, for instance those representing routes in the Capacitated vehicle routing problem, where the depot is represented multiple times. Specifically, the number of genes in the resulting individuals representing depots changes with each crossover, until the population consists of individuals with only depots in them. Problems involving multiple instances of the depot gene do not use this operator.

The algorithm implemented for MSTPP proved to work well, as intended. Shipments' routes cluster around carriers that are beneficial for carrying many shipments at once. Still shipments do not get scheduled for any major detours since it would affect the overall cost.

Initially we thought that the given cost structure would raise problems. Most papers and researches found handles costs of full vehicles or conveyances where we wanted to be able to buy space of another company owning the vehicle. It turned out with some minor modifications that we could use the proposed algorithm for MSTPP with our cost structure. As already mentioned the situation of buying space from a vehicle is not applicable in VRP since the vehicle must be scheduled fully by our self, hence the whole vehicle is hired or owned.

By combining the solved sub problems into a single system we managed to get a working optimization module that schedules orders from source address to destination address involving pick-up, cross-dock routing and delivery.

References

[1] Google Code. Google web tools. `http://code.google.com/webtoolkit/`, August 2011.

[2] B. D. Díaz. The vrp web. `http://neo.lcc.uma.es/radi-aeb/WebVRP/`, March 2007. viewed on 25 August 2011.

[3] D. Eppstein. Finding the k shortest paths. *Foundations of Computer Science*, 35th Annual Symposium on Foundations of Computer Science:154–165, November 1994.

[4] S. Bengio J-Y. Potvin. The vehicle routing problem with time windows - part ii: Genetic search. *INFORMS Journal on Computing*, volume 8:165–172, 1996.

[5] B. Shen L. Lin, J. Hu. A new hybrid method of genetic algorithm,tabu and chaotic search for cvrptw. *Intelligent Computing and Intelligent Systems (ICIS)*, 2010 IEEE International Conference on, vol 2:336–340, October 2010.

[6] H. Okano M. Amano, T. Yoshizumi. The modal-shift transportation planning problem and its fast steepest descent algorithm. *Simulation Conference, Proceedings of the 2003 Winter Simulation Conference*, pages 1720–1728, December 2003.

[7] M. M. Solomon. Vrptw benchmark problems. `http://w.cba.neu.edu/~msolomon/problems.htm`. viewed on 10 November 2011.

[8] M. M. Solomon. Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations Research*, Volume 35, No. 2:254–265, 1987.

[9] T. Starkweather, S. McDaniel, K. Mathias, D. Whitley, and C. Whitley. A comparison of genetic sequencing operators. *Proceedings of the fourth International Conference on Genetic Algorithms*, pages 69–76, 1991.

[10] A. Win. The value a 4pl provider can contribute to an organisation. *International Journal of Physical Distribution & Logistics Management*, volume 38 issue 9:674–684, 2008.

[11] David Yu. Protostuff. `http://code.google.com/p/protostuff/`. viewed on 16 November 2011.