Master's degree thesis

LOG950 Logistics

Comparison of Metaheuristics for the Single Vehicle Pickup and Delivery Routing Problem with Multiple Commodities

Siarhei Tsitou

Number of pages included the first page: 79

Molde, 25 May 2009

Molde University College

Publication agreement

Title: Comparison of Metaheuristics for the Single Vehicle Pickup and Delivery Routing Problem with Multiple Commodities

Author(s): Siarhei Tsitou

Subject code: LOG950

ECTS credits: 30

Year: 2009

Supervisor: Irina Gribkovskaia

Acknowledgements

I would like to thank my relatives for their kind patience and support during my work on this thesis. I would also like to thank Professor Irina Gribkovskaia, Høgskolen i Molde, for her valuable advices and support, Associate Professor Øyvind Halskau, Høgskolen i Molde, for his remarks and suggestions, Professor Gilbert Laporte, HEC Montréal, for inspiring and directing me on the right path, and Christopher Clark for providing a hashtable implementation in C programming language.

Abstract

The problem that we are going to study in this thesis is an extension of a Vehicle Routing Problem with Pickups and Deliveries - *Single Vehicle Pickup and Delivery Problem with Multiple Commodities*. In this problem there is only one vehicle that serves customers which have delivery and pickup demands in many commodities. The main objective of the problem is to find the least cost route, provided that pickup and delivery demands of all customers are satisfied. Additionally, the vehicle capacity should not be exceeded for any of the commodities. At present only one method which allows for solving Single Vehicle Pickup and Delivery Problem with Multiple Commodities exists – it is a heuristic by Gjengstrø and Vaksvik (2008). So, there is the possibility for improvements.

This thesis presents two new Tabu search based metaheuristics. These metaheuristics are more advanced in comparison to heuristic by Gjengstrø and Vaksvik. Their core concept is changing the neighborhood when the search process sticks in the local optimum and can no longer improve the solution. The first metaheuristic utilizes the idea of Variable Neighborhood Search and changes the neighborhood each time the current solution cannot be improved. The second is more sophisticated – it selects the neighborhood with the probability which is calculated according to the search history. It also uses Simulated annealing technique to accept worse solutions. Computational results show that both metaheuristics are especially good in solving problem instances where the vehicle load is less than 100% of its maximal capacity. It is also shown that changing the load influences shapes of the obtained solutions and the number of visits to customers in the routes. Assumption that optimal solutions have only customers that are visited once is not true. We present situations when Hamiltonian-shaped routes are load-infeasible and the optimal solution then is non-Hamiltonian. Sometimes even if Hamiltonian solutions are feasible the route of the least cost is non-Hamiltonian and thus, it has customers which are visited more than once.

Keywords: single vehicle; pickup and delivery problems; multiple commodities; tabu search; variable neighborhood selection; probabilistic neighborhood selection.

Contents

1 Introduction

Single Vehicle Pickup and Delivery Problem with Multiple Commodities belongs to a group of Vehicle Routing Problems (VRPs). This group includes such problems as Capacitated and Distance-Constrained VRP, VRP with Time windows, and VRP with Pickups and Deliveries (VRPPD). We are especially interested in the latter one.

VRPPD is a routing problem with a depot and a set of customers. There is a fleet of vehicles of some load capacity that start at the depot, visit all customers and return back to the depot. When a vehicle visits a customer it delivers some commodities and picks up the other. Deliveries can be performed separately from pickups. One must admit that in some problem extensions it can be stated additionally that deliveries should necessarily precede pickups, like it is done in the case of VRP with Backhauls, for instance, – we will talk about this problem later. In VRPPD we do not have such restriction. Hence, pickups and deliveries can be performed in any order. VRPPD's objective is to find a route of the minimal cost. For such route all customers' demands should be satisfied and vehicles' capacities should not be violated in any period of time.

VRPs can have one vehicle that serves all customers or several vehicles. Customers can have demands in multiple commodities or only in one commodity.

If for VRP with pickup and deliveries we additionally assume that only one vehicle is available for serving customers and each customer has pickup and delivery demands not in one but in a number of commodities, then we get a Single Vehicle Pickup and Delivery Problem with Multiple Commodities (SVPDPMC). This is the problem we are going to study in this thesis.

For simplicity we assume that split deliveries and pickups are not allowed. To split a delivery or a pickup means to perform it not in one but in several visits. So, we assume that deliveries and pickups should be finished in exactly the same visits when they start.

SVPDPMC has many real life applications. For instance, this kind of problem may arise in oil and gas industry when a vessel has to serve the offshore platforms. In this case we have two types of commodities: bulk and deck. Each type of commodity has its own maximal allowed capacity. Bulk commodities can be subdivided into families – groups of products which can be transported in the same transportation tanks. They include, for instance, water, diesel oil, cement, and methanol. Deck commodities are transported on a vessel deck. They can also be stored in different transportation units: boxes, tanks, and containers. The example of deck commodities are food, materials, and equipment. All commodities are delivered to the offshore locations from the depot – supply base. Some of them should be picked up at the offshore platforms and delivered back to the depot. Platforms represent customers with pickup and delivery demands. Base represents depot. More information about deck and bulk commodities can be found in Zouari and Akselvoll (2007).

SVPDPMC is the NP-hard problem, i.e. there is no algorithm that can solves this problem in a polynomial time. Exact methods need too much time to find a solution, so then can be scarcely used in real life problem applications. In the cases when near-optimal solutions are allowed one can use heuristics. To our best knowledge only one such algorithm for SVPDPMC exists today. This is the heuristic by Gjengstrø and Vaksvik (2008). In this work Gjengstrø and Vaksvik also solved the problem exactly with CPLEX. They found out that CPLEX failed to find solutions for some problem instances in the given limit of time. Additionally, not on all instances their heuristic was able to obtain best-known solutions. So, there is the great opportunity for improvements.

In this work we present two metaheuristic algorithms that solve SVPDPMC. The first algorithm incorporates the idea of Variable Neighborhood Search and changes the neighborhood each time the heuristic fails to find the better solution. The second algorithm goes further and uses probabilistic neighborhood selection. It changes the neighborhood in accordance with the probability which depends on the solution search history. It can also accept worse solutions with the help of Simulated annealing technique.

Each of these two metaheuristics has its specific features and may perform better as well as worse depending on a number of factors such as vehicle load and number of commodities.

We are going to show that solution shapes correlate with the vehicle load. Not only solution costs will be compared but also CPU time needed to achieve these solutions. Tests will show us that two our new algorithms need less time to complete 100000 iterations than heuristic by Gjengstrø and Vaksvik. Smaller CPU time values can give the advantage in solving bigger problem instances and in performing more algorithm iterations in a fixed time. This in perspective allows for finding better solutions. Finally, to reveal how close we are to the best known solutions we compare our results with the results obtained by Gjengstrø and Vaksvik on their AMPL model.

This thesis is organized as follows. First of all we discuss combinatorial optimization problems and two types of methods they can be solved with. In the next section we introduce SVPDPMC as one of the extensions for the VRP, present methods for solving it and discuss solution shapes one may obtain. In section 4 we start from general concepts of

Tabu search and finish with the description of two metaheuristics we are going to use. In section 5 we present computational results of metaheuristics from the previous section and discuss them. Finally, in two last sections we make conclusions and outline possible directions for the further research.

2 Combinatorial Optimization

SVPDPMC belongs to combinatorial optimization problems. These are the problems which feasible solutions are either discrete or can be reduced to discrete ones. Their goal is to find the best of all feasible solutions. The examples of such problems are: Travelling Salesman Problem, Minimum Spanning Tree, Knapsack Problem, Cutting Stock Problem, Eight Queens' Puzzle and VRP. We are going to deal with only VRP and its extensions. Descriptions of other combinatorial problems one can find in literature, for example in Kellerer et al. (2004) or in Baranov et al. (1995).

Before describing our metaheuristics we would like to list possible approaches how combinatorial optimization problems in general and our VRP extension as one of such problems can be solved.

2.1 *Solving Methods*

For solving combinatorial optimization problems one may use two types of methods. Each of these types has its strong and weak sides. In general one cannot assert superiority of one method over the other, for on different problem instances distinct methods perform differently.

2.1.1 Exact Methods

The most well-known exact methods are: Branch and Bound, Branch and Cut, Cutting planes algorithm, and Brute-force search. The first two methods are well described by Chinneck (2007), the third method is discussed later in this thesis and the fourth method is a very simple but also a very inefficient method where all possible solutions are enumerated and checked for satisfaction of the problem objectives.

The advantage of using exact methods is that such method always gives the globally optimal solution. The disadvantage is time inefficiency.

2.1.2 Heuristics, Metaheuristics and Hyper-heuristics

Unlike exact methods heuristics and metaheuristics do not provide feasible solutions which are proved to be optimal in a global sense. Instead, they provide pretty good ones but in the less time.

One of the bad things about heuristics and metaheuristics is that they can get into trouble with local optimums. It means that they can reach solutions which are locally optimal and then stick and not be able to find the better. The good thing about them is that they can be applied to problems for which no other methods are known.

Whereas metaheuristic means the higher level or advanced heuristics, hyper-heuristic is a heuristic of choosing or constructing a heuristic. The reason of it existence is based on the fact that, as it has already been said before, all methods perform differently on different problem instances. Thus, for instance, having heuristic H1, heuristic H2, and two problems: P1 and P2; applying H1 to P1 and to P2, H2 to P1 and to P2 one may encounter the situation when H1 on P1 performs better than H2 on P1 but worse on P2 than H2. In such situation one may talk about heuristic procedure able to select heuristic that produces better results taking into account the problem special features, and such heuristic procedure is called a Hyper-heuristic.

Hyper-heuristics are successfully used when solving different combinatorial problems as, for instance, VRP. They can be divided into two main categories: heuristics to generate heuristics and heuristics to choose heuristics. Heuristics to generate heuristics generates new heuristics from available heuristic components and heuristics to choose heuristics chooses heuristics from the set of already known methods.

Besides using heuristics themselves one can use their combinations. If a heuristic uses some other predefined heuristics to make itself more efficient, such heuristic is usually called a metaheuristics. In our work we are going to combine Tabu search, Sweep algorithm and Simulated annealing to develop two new metaheuristics.

The following list shows the most common methods that are used to solve combinatorial optimization problems so far.

- Ant colony optimization
- Bees algorithm
- Clarke and Wright savings heuristic
- Fisher and Jaikumar heuristic
- Genetic algorithms

- GRASP
- Hopfield net
- Local search and Generalized local search
- Nearest neighbor heuristic
- Quantum annealing
- Reactive search
- Self-organizing map
- Simulated annealing
- Swarm intelligence
- Sweep algorithm
- Tabu search

3 Vehicle Routing Problems

The problem we study in out master thesis belongs to a VRP class. VRP is a combinatorial optimization problem which is formulated as follows. A fleet of vehicles is located at the depot where it starts from. This fleet visits all customers, performs their service and returns back to the depot. Each customer has a demand. When a vehicle delivers products to customers it satisfies their demands. The objective of VRP is to find the route of the minimal length (or cost), provided that demands of all customers are satisfied. There are also restrictions on vehicles' load capacity.

The next list contains the main VRP extensions.

- VRP with time windows (VRPTW) is a VRP where additional constraints on customers' service times are implied. Service time intervals that state when customers can be serviced by vehicles are usually called Time windows. When a vehicle comes to the customer to serve it three possible situations are possible. First, the vehicle can arrive earlier, i.e. before the beginning of the allowed time window. Then it has to wait until it can start performing the service. Second, the vehicle can arrive in time, and then it just serves the customer and leaves. It is easy to see that it may also leave as during the time window interval so as after it ends, depending on the time needed to serve the customer. In any case it does not violate any constraints. Finally, the vehicle can be late. In this case one may talk about soft constrained problem where a penalty for lateness is paid, or about hard constrained problem where such lateness is banned at all.
- Capacitated VRP or CVRP. This kind of problem arises when vehicles have limited capacity for storing goods that are to be delivered and, thus, vehicles may need to return to the central depot for replenishment. CVRP can be used with time windows constraints and then, sure enough, they are called CVRP with Time windows.
- Distance-constrained VRPs. These are VRPs where the travelling time and/or distance is limited. It can be due to the limit in the driver's working day or in a travel distance on one full fuel tank.
- VRP with Multiple depots. Vehicles can be scheduled from multiple depots. The goal is to build the optimal schedule.
- VRP with Pickups and Deliveries or shortly (PDPs) is a type of VRP where each customer has a demand for products to be picked up as well as a demand for

products to be delivered. Thus, a fleet of vehicles should not only deliver products to their destinations but also pick up some other products at these locations and bring them back to the depot. It is obvious that in a particular case when pickup demands are equal to zeroes we get a simple VRP. One may encounter a problem instance where the delivery and/or pickup demands are equal to zeroes only for a selected set of customers.

3.1 Vehicle Routing Problems with Pickups and Deliveries

Barbeglia at al. (2007) suggests subdividing PDPs into classes by structure, visits and number of vehicles. So, we can name three classes basing on the relation between commodities' sources and destinations.

- One-to-One PDP (1-1 PDP). This is a class of pickup and delivery problems where there exists a one-to-one correspondence between collected and delivered products, i.e. for each commodity from a certain location there is only one delivery destination. Dial-a-ride PDP mentioned further is the example of such a problem.
- Many-to-many PDP (M-M PDP) modification where vertices can be sources or destinations for any commodities, i.e. commodities can be collected from different locations and then delivered to customers which have demand in such commodities. The good example, however, that has one commodity is given by Zouari and Akselvoll (2008) where the milk is picked up at cow farms and supplied to private residents. The more complicated case arises when several commodities are involved, for instance, milk and honey. Milk can be picked up from milk farms, honey can be picked up from bee-gardens and they are both destined to the set of customers.
- One-to-Many-to-One PDP (1-M-1 PDP). These PDPs have one depot which is the source of commodities for customers but this depot is also the destination for the commodities that are picked up at the customer locations.

Subdividing by visits comes out of the possibility to perform pickups and deliveries in different visits to customers. We may perform pickups and deliveries at the same time in one visit – such VRPs are called VRPs with simultaneous services. Solutions to these problems are Hamiltonian cycles. One may think that the optimal solution of any minimization combinatorial problem can always be found among feasible

Hamiltonian-shaped solutions but it is not always true. We will show further that in some cases non-Hamiltonian shaped solutions can give routes of the less cost. Customers can also be serviced in different visits: for example, delivery can be made in the first visit to the customer and pickup – in the second visit. For some nodes we may perform pickup and delivery at the same time. Solutions in this case will have non-Hamiltonian shapes.

This classification does not define the whole spectrum of possible pickup and delivery problems with their extensions. Nevertheless, it is useful and we will utilize it in this thesis.

The following is the list of the most common PDP extensions.

- Traveling salesman problem with pickup and deliveries with first-in-first-out loading where deliveries and pickups are executed according to the order of precedence. This is a 1-M-1 class of problems, using Barbeglia at al. notation. The main objective of this problem is to find a least cost route where all customers' requests are fulfilled. Each customer's request is associated with pickup location and delivery location. So, the vehicle should pickup something at the first location and then it should deliver it to the second location. The vehicle starts from the depot and returns back after fulfilling the requests. This problem is discussed by Erdoğan at al. (2007).
- PDP with time windows. These are PDPs where customers have desired service time windows. If a vehicle arrives before the time window it should wait to perform a service. If it arrives late, then it can be penalized with some extra cost. Arriving late can be prohibited. The first situation gives us a soft constraint and the second – a hard constraint. Soft and hard constraints have been already mentioned before in this thesis. This is a 1-M-1 problem.
- PDP with two-dimensional loading constraint (2L-PDP). This problem is a PDP special case where additional constraint on maximal available height and weight is imposed. In this problem we consider that every product has its own size and weight. All products loaded into the vehicle should not exceed the maximal allowed height, width and weight. Thus this problem combines a VRP together with a packing problem. This type of problem is studied by Malapert at al. (2008). 2L-PDP belongs to 1-M-1 problems.
- PDP with multiple depots. This is the type of problem where vehicles are located at different depots. The objective is to select such depots that customers will be

served in the optimal way, say, for example, in a minimal amount of time or with the least cost. Depots may also be used for intermediate replenishments of vehicles along their route. Such problem is called a Multi-depot VRP with Inter-depot Routes. It is discussed by Crevier at al. (2007). This is a M-M PDP.

- PDP with route duration or total cost constraints. It is a basic PDP but with the addition that cost and/or route duration is limited to some value.
- Stacker Crane Problem (SCP) is a problem where one server should fulfill all requests. Each request is connected with the delivery of an item or a group of items from the source to the destination. The goal is to fulfill all requests in such sequence that the length of the route will be minimal. This kind of problems was described by Coja-Oghlan at al. (2003). According to Barbeglia at al. (2007) it is a 1-1 problem.
- Dial-a-ride PDP (DARP) is a problem where a number of vehicles serve all requests in the optimal way. This problem is an extension of the SCP and is also a one-to-one PDP.
- PDPs with selective pickups. The main difference from the pure PDP is that it is not necessary to satisfy all pickup demands. Such problem is studied by Gribkovskaia at al. (2008) and is called as Single VRP with Deliveries and Selective Pickups Problem (SVRPDSP). In SVRPDSP pickup revenue is associated with each vertex and thus, a pickup demand is satisfied only for those vertices that give profit. This problem belongs to 1-M-1 class of PDPs.

3.2 Single Vehicle Pickup and Delivery Problem with Multiple Commodities

3.2.1 Problem Definition

According to Barbeglia at al. (2007) SVPDPMC is the M-1-M PDP with multiple visits to customers and one vehicle. To be precise, one can formulate this problem as follows.

Let $G = (V, A)$ be a graph, where $V = \{0,1,2,\dots,n\}, n \in \boldsymbol{N}$, is a vertex set, and $A = \{(i,j) \mid \forall i,j \in V, i \neq j\}$ is an arc set. Without loss of generality let's assume that 0 is a depot. Let n be the maximal number of available customers. With each arc $(i,j), \forall i,j \in$

$V, i \neq j$ of such graph G a non negative cost c_{ij} is put into correspondence. c_{ij} is usually referenced as a cost of traveling from vertex i to vertex j.

Now let's assume we have a set of commodities $K = \{1, \dots, k\}$. For each vertex $i, i \neq 0$ of graph G the pair of values (d_{ih}, p_{ik}), $d_{ih} \geq 0$, $p_{ik} \geq 0, i \in V\backslash\{0\}, h, k \in K$ is known. d_{ih} is a demand of a vertex i in a commodity h, and p_{ik} is the amount of a commodity k to be picked up from a vertex i. The solution for SVPDPMC is a set of $m > 0$ routes that start and end at the vertex 0 and cover all vertices from $V\backslash\{0\}$. The sum of all routes' costs is subject to minimization. It is easy to see that in case if $m = 1$ we may get the least cost Hamiltonian cycle but according to Gribkovskaia at al. (2007) it happens very rare. The capacity of a vehicle is Q.

Loading constraints give us two possible variants.

First variant is used when we do not differentiate between storage conditions of commodities of different types. For instance, if we deliver bottles with soda and beer, bottles can be stored in the same boxes and located in the same conditions in a truck. Therefore, the capacity limit will be the same for both products.

Then, according to Gribkovskaia at al. (2007) to be able to find a feasible solution we should be aware that the following inequalities hold:

$$\sum_{i=1}^{n} \sum_{h=1}^{k} d_{ih} \leq Q \,, \text{ and } \sum_{i=1}^{n} \sum_{h=1}^{k} p_{ih} \leq Q$$

The second variant is used when commodities are stored and transported in different conditions. The example: transportation of frozen beef and beer. Frozen beef is stored in fridges, whereas beer is transported in boxes in normal temperature. So, storage limit for beer and beef will be different. Another example: deck and bulk commodities on a vessel. In general case we get the following inequalities.

$$\sum_{i=1}^{n} d_{ih} \leq Q_h, h = \overline{1, k}, \text{ and } \sum_{i=1}^{n} p_{ih} \leq Q_h, h = \overline{1, k}$$

3.2.2 Literature Review

Vehicle routing problems have been studied by many authors. A lot of scientific papers devoted to VRP and its extensions are published every year. One can find a lot of information about VRPs and its extensions for example in Toth and Vigo (1987).

Nevertheless, to our knowledge only two works were devoted to SVPDPMC: Zouari and Akselvoll (2007), and Gjengstrø and Vaksvik (2008). Zouari and Akselvoll built and described a deterministic model for SVPDPMC with Time windows, Gjengstrø and Vaksvik presented CPLEX program for solving SVPDPMC with and without Time windows exactly and also a tabu-based heuristic. In our Master thesis we are going to use the results by Gjengstrø and Vaksvik.

3.2.3 Research Purpose

The purpose of this Master thesis is to develop, to describe, to implement and to test two metaheuristic algorithms for solving SVPDPMC. These metaheuristics are based on Unified Tabu Search Heuristics Algorithm (UTSA) by Jean-François Cordeau, Gilbert Laporte and Anne Mercier (2001) and also its modification by Gjengstø and Vaksvik. Two implemented algorithms will be tested on the set of instances from Gjengstø and Vaksvik (2008). Then these metaheuristics will be compared with each other and the already existing single heuristic method from the latter work.

3.2.4 Problem Solution Shapes

One of the objectives of our work is to find out how a vehicle load influences problem solution shapes. Solution shapes can be of two types: non-Hamiltonian and Hamiltonian. In Hamiltonian solutions all customers are visited only once and in non-Hamiltonian solutions at least one customer is visited more than once.
To demonstrate possible SVPDPMC solution shapes we assume the following simple problem instance. Single depot with one vehicle should serve 4 customers: customer 1, customer 2, customer 3, and customer 4. Below one can find two tables with travel costs and customers' demands we use in our example.
Travel costs satisfy a triangle inequality: $c_{ij} \leq c_{ik} + c_{kj}$, where i, j, k – three incidental vertices which represent customers.

Travel costs					
Customers	depot	1	2	3	4
depot	-	7	13	20	19
1	7	-	9	15	17
2	13	9	-	13	20
3	20	15	13	-	17
4	19	17	20	17	-

Table 1: Costs of traveling between vertices

Demands (pickup, delivery)					
Commodities \ Customers	depot	1	2	3	4
Commodity 1	-	(2,1)	(2,3)	(1,2)	(3,2)
Commodity 2	-	(3,3)	(3,1)	(2,1)	(2,2)
Commodity 1 load limit = 8 Commodity 2 load limit = 10					

Table 2: Pickup and delivery demands

Double path

The simplest method that one can use to find a feasible route is a double path. This technique always gives a feasible but not necessarily optimal solution. Double path solution can be constructed as follows: vehicle starts from the depot and simultaneously visits all unvisited nodes. After all unvisited nodes are visited and serviced the vehicle returns to the depot exactly the same way as it arrived. In the first visit to the node the vehicle accomplishes deliveries and in the second visit it accomplishes pickups. The last unvisited vertex in a route is visited only once, hence, for this node both delivery and pickup are accomplished at once. The next figure shows solution shape that one gets using double path.

The blue rectangle represents a depot and the circles with numbers are customers. “P” means that the vehicle accomplishes pickup, and “D” means that it accomplishes delivery; “PD” means that both services are performed.

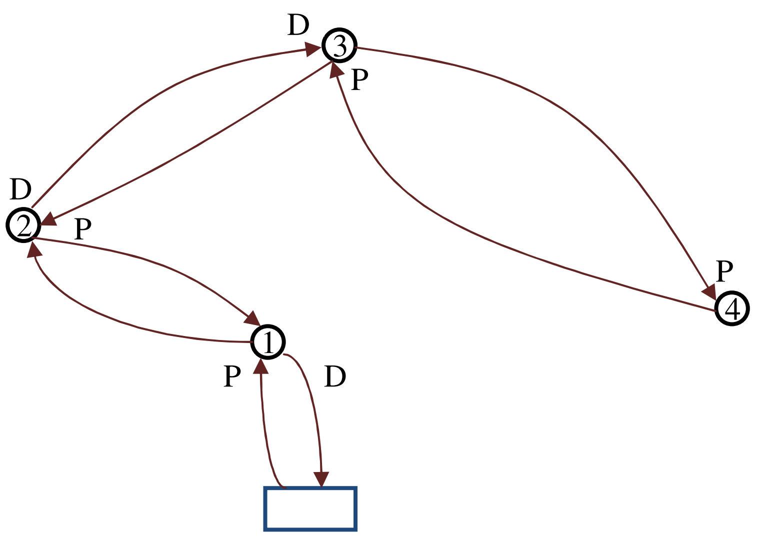

Figure 1: Double path route

Travel cost of this route is: 92.

Non-Hamiltonian cycle

This shape gives the better solution than double path in the previous section. Here we got a so-called lasso solution which can be seen in Figure 2. In general not all non-Hamiltonian solutions are lasso solutions.

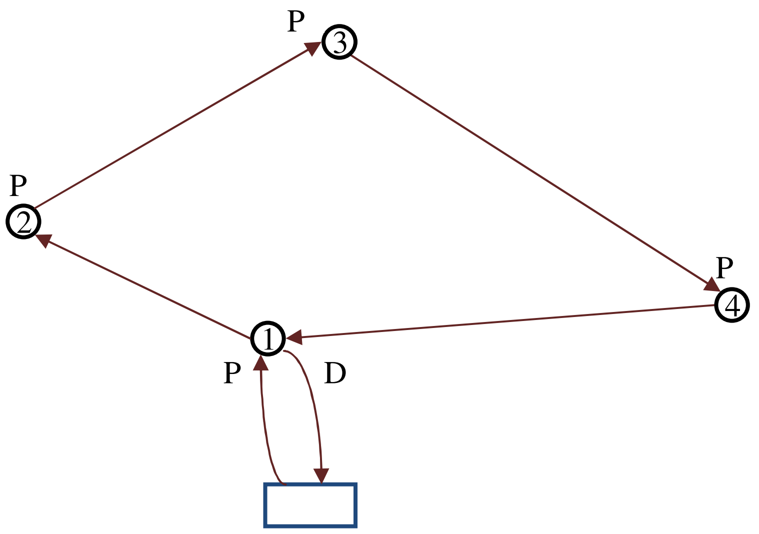

Figure 2: Non-Hamiltonian-shaped solution

Travel cost of this route is: 70.

Hamiltonian cycle

In our case connected graphs which vertices have degrees equal to 2 represent Hamiltonian-shaped solutions.

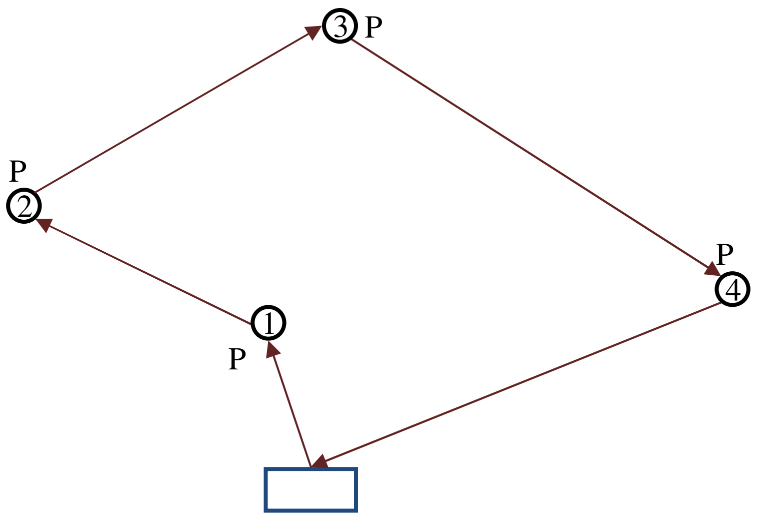

Figure 3: Hamiltonian-shaped solution

Travel cost of this route is: 65.

In this particular case Hamiltonian shape gives the minimal travel cost. This is the best possible solution for this problem instance.

However, it is not always possible to find a feasible solution which is Hamiltonian-shaped. Moreover, sometimes even feasible Hamiltonian-shaped solutions have higher costs than non-Hamiltonian solutions. To demonstrate this we present two simple examples: when there is no feasible Hamiltonian-shaped solution, and when such solution exists but it is not optimal.

Let's look on the situation when no feasible Hamiltonian cycles exist. We modify pickup and delivery demands for both commodities in our sample problem instance as follows.

Demands (pickup, delivery)					
Commodities \ Customers	depot	1	2	3	4
Commodity 1	-	(2,1)	(2,3)	(1,2)	(3,2)
Commodity 2	-	(2,3)	(3,1)	(3,1)	(2,4)
Commodity 1 load limit = 8 Commodity 2 load limit = 10					

Table 3: There is no feasible Hamiltonian-shaped solution

In this case it is impossible to satisfy the demand of vertices 1 and 4 in commodity 1, and demand of vertices 2 and 3 in commodity 2 in only one visit. Thus, at least one of these vertices should be visited twice. So, Hamiltonian cycle cannot give a feasible solution.

Generally speaking, it can be impossible to construct feasible Hamiltonian-shaped solutions not only because such solutions can be load-infeasible but also because they can be infeasible due to violation of time windows constraint.

In this example the optimal solution has non-Hamiltonian shape. This solution is shown in the Figure 4.

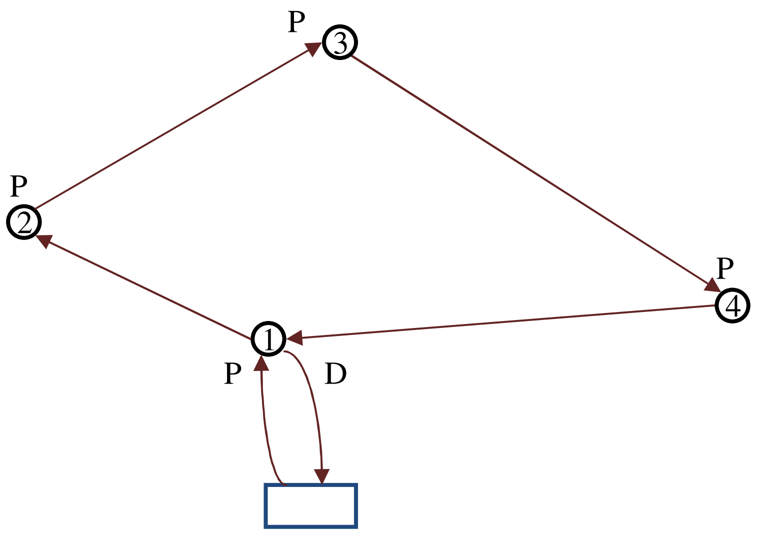

Figure 4: Optimal but not Hamiltonian-shaped solution

Travel cost of this route is: 70

In this optimal solution delivery and pickup are separated for vertex 1. It allows load constraint satisfaction. So, in the first visit to vertex 1 the vehicle performs deliveries and in the second visit it performs pickups. Table 4 shows the load of the vehicle and its activity while it travels along the route. "P" means pickup and "D" means delivery.

Origin – destination	**Commodity 1 load (limit=8)**	**Commodity 2 load (limit=10)**	**Commodity 1 destination service**	**Commodity 2 destination service**
Depot - 1	8	9	D	D
1 - 2	7	6	PD	PD
2 - 3	6	8	PD	PD
3 - 4	5	10	PD	PD
4 - 1	6	8	P	P
1 - Depot	8	10	-	-

Table 4: Vehicle load level and service activity along the route

The next situation is possible when a feasible Hamiltonian cycle solution does exist but it is not optimal. For brevity we take only one commodity. In our problem instance we should change commodity 1 pickup and delivery. The next table illustrates changed instance data.

Demands (pickup, delivery)					
Commodities \ Customers	depot	1	2	3	4
Commodity 1	-	(3,1)	(2,3)	(1,2)	(3,2)
Commodity 2	-	-	-	-	-
Commodity 1 load limit = 8 Commodity 2 load limit = 0					

Table 5: Hamiltonian-shaped solution exists but it is not optimal

The best feasible Hamiltonian-shaped solution gives the route cost equal to 73 and this route is shown in Figure 5.

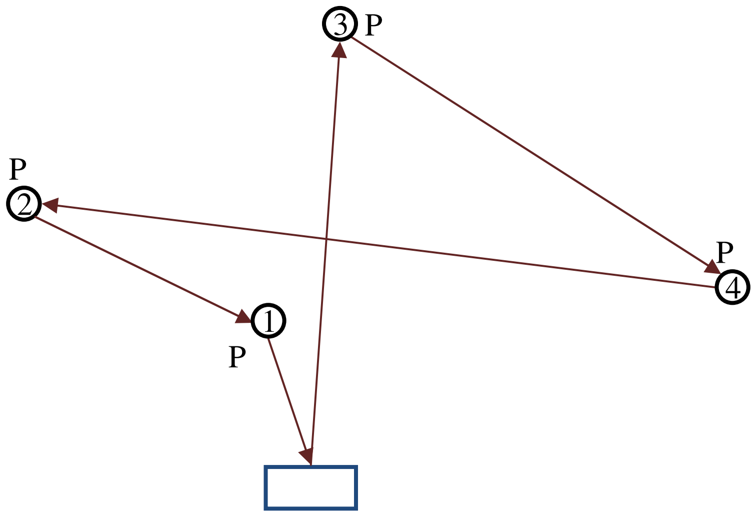

Figure 5: The best feasible Hamiltonian-shaped solution

The optimal non-Hamiltonian (in this case it is a lasso) solution gives the route with the cost equal to 70 < 73.

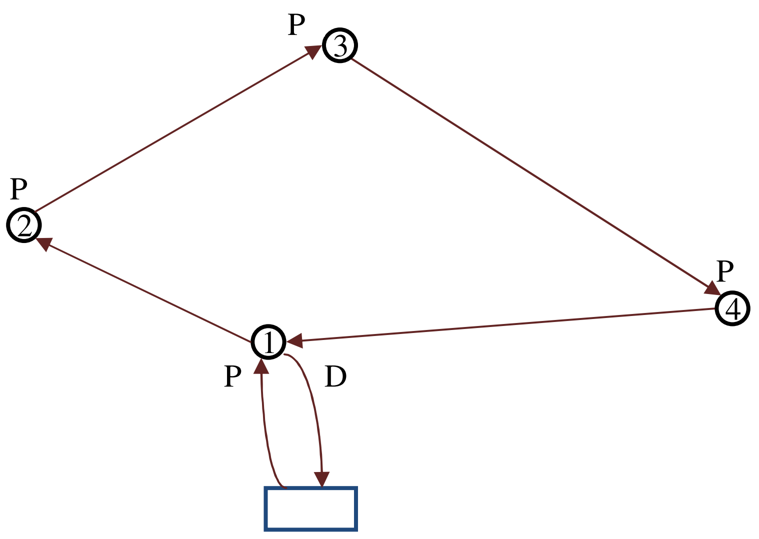

Figure 6: The optimal lasso-solution

By these examples we outlined different situations that may appear when building a solution for SVPDPMC, namely:

- Optimal solution is a Hamiltonian cycle
- All Hamiltonian-shaped solutions are infeasible. Therefore, the optimal solution is non-Hamiltonian-shaped
- Feasible Hamiltonian-shaped solutions exist but the optimal one is non-Hamiltonian-shaped

3.2.5 Solving Methods

As for any of VRPs to solve SVPDPMC one can use exact methods.

For example we can use a Cutting plane algorithm. The algorithm starts from solving the linear programming (LP) relaxation. After getting a solution it checks its integrality. If integrality constraints hold then we state that the found solution is the optimal and algorithm stops. Otherwise, the algorithm generates cutting planes and cuts for the LP relaxation and tries to solve the problem once more; it repeats these step until the optimal solution can be found. For SVPDPMC problem it may look as follows:

(a) In the SVPDPMC model some of its constraints are relaxed, say it is a cost feasibility constraint

(b) Solution for the relaxed problem is constructed

(c) We find out where exactly in a route the relaxed cost feasibility constraint is violated. Then we insert this constraint for particularly that place in a route where it is violated. This represents the cutting plane for the LP relaxation.

(d) Algorithm is repeated from the step (b) until everything is satisfied and the optimal solution is found

Cutting planes can be generated from the conjunction but according to Ceria (1998) they may be generated from some other logical conditions.

We can also solve the problem with the already mentioned methods as Branch and Bound, Branch and Cut, and Brute-force.

Finally, one can use heuristic methods such as Tabu search, Simulated annealing or Genetic algorithms.

4 Tabu Search Algorithm

SVPDPMC is the NP-hard problem, so exact methods like Cutting planes algorithm will often need a lot of time to solve a real-life problem instance even of a medium size. Time waste can be avoided by using heuristics in cases when the near-optimal solutions are accepted. One of such heuristics is a Tabu search (TS).

TS was firstly described by Glover in 1986 and since then it was used by many people for solving lots of different problems.

4.1 Concept of Tabu Search

According to Oklobdzija (2002) TS is a metaheuristics that can be superimposed on any algorithmic method if this method constructs new solutions from already existing solutions by applying a sequence of moves. Such moves can be of a different nature: adding or removing a vertex from a route, adding or removing an item from a knapsack, etc. What will be considered as a move depends on a particular problem instance and on a context in which it is used. To avoid cycling TS uses tabu tenure – restriction of some moves for a number of algorithm iterations. Then one says that these moves are declared tabu. Tabu tenure can be either fixed or dynamic. A fixed tabu tenure mean that moves are always penalized for a predefined number of iterations and dynamic tabu tenure means that this number of iterations changes while algorithm runs. According to Glover and Laguna (1997) dynamic tabu tenure can be changed every selected number of iterations during which it remains unchanged. It can also be changed each time some attribute becomes tabu.

One of the TS core elements is a solution neighborhood which is a set of all possible solutions reachable from the currently available one by applying a set of moves.

4.1.1 Termination Criteria

The following termination criteria can be used with TS. So, TS stops when:

- the iterations limit is reached
- the specified objective function value is reached
- solution is not improved for a particular number of iterations
- the algorithm running time limit is reached

4.1.2 Aspiration Criteria

Aspiration Criteria is one of the most important concepts of TS. Due to the nature of the TS it can happen that the search process is not able to reach the better possible solutions since moves that lead to such solutions are prohibited. Therefore, one must use some algorithmic procedure to make these moves not tabu. These algorithmic procedures are called aspiration criteria procedures.

The mostly used aspiration criterion is allowing previously restricted moves that give solutions with better objective function values than the value of the best known solution so far. However there are some other methods but they are not so widely implemented.

4.1.3 Diversification

Diversification technique makes TS very powerful. It diversifies the search process and helps it to move to the new regions where possibly better solutions can be found. If a solution space is very volatile – has a lot of local optimums, then diversification is especially helpful because it helps to overcome peaks and troughs in the solution space.

There are several approaches for implementation of the diversification.

The first approach is to use a *restart method*. The method consists in applying rarely used attributes to the current or best known solution and restarting the search process.

The second approach is to use a *continuous diversification*. This method diversifies the search process when the algorithm runs. As Klein (2000) states one can use a frequency based memory to continuously diversify the search. Using information provided by this frequency based memory one can ban attributes that were frequently used during the search process for a number of iterations. This will lead to diversification, for rarely used attributes will be used more often and thus new solutions will be explored.

4.1.4 Intensification

Intensification is not the essential part of the TS and it can be omitted in its implementation. Intensification forces the search process to seek solutions in the most promising areas of the search space.

Intensification can be implemented by means of path relinking that we will talk about later.

Another approach is to use *decomposition*: the search region is restricted by imposing additional constraints on the initial problem. This means that some of the problem variables become fixed and the search is performed for the rest of variables.

4.1.5 Other Methods and Strategies

One may also use *strategic oscillation.* The main idea is to allow the search process to accept infeasible solutions. The objective function must then incorporate weighted penalties for constraints violation. During the algorithm run these weights are adjusted according to the current search history: increasing weights for infeasible solutions and decreasing for feasible.

Elite Configurations are the best solutions configurations that were found during the search. Lists with such elite solutions can be used in the process restart for diversification, intensification or it can be used in path relinking.

Path relinking can be used for intensification and / or for diversification. Path relinking produces new solutions from elite solutions, for instance by seeking new solutions in the neighborhood of elite solutions. The other possibility is to use the parts of elite solutions to construct new solutions. The more parts from the elite solutions are taken – the more intensified the search is; the less parts of the elite solutions are taken – the more diversified the search is.

More detailed information about TS and its strategies one can found in Glover and Laguna (1997), or Lee at al. (2008).

4.1.6 Surrogate and Auxiliary Objective Functions

For a great many of problems it is not cost efficient to calculate the real objective functions values. This may lead to computing overheads and slow algorithm run. One can imagine for example the situation when each solution in the neighborhood is evaluated by computing the objective function this solution realizes. Then, for a chosen solution with the best objective function value its own neighborhood should be explored. Finally, objective function values should be recalculated for all solutions in this new neighborhood. After this the process repeats. To avoid such heavy calculations *surrogate objectives* can be used. Surrogate objective is a function that correlates to the real objective function but is easier to maintain and to compute.

One more reason not to use the real objective function is that it may lack information that can be useful to guide the search process to the more promising regions. In such case *auxiliary objective function* is used which incorporates solutions' desired attributes. Such attributes depend on a particular problem and for instance it can be load feasibility or time windows feasibility.

Surrogate and auxiliary objective functions are widely used due to their efficiency.

4.1.7 Simple Tabu Search Template

TS concept functioning can be made easier to understand with the following simple TS template. It shows only the basic structure of the TS which, however, in practice can include different tricky techniques to make the search more efficient.

Let $f(s)$ be the function to be minimized, s – a solution from the search space which fitness can be evaluated with this function.

Notation

s – current problem solution

s^* – best known solution

s^0 – initial solution

$N(s)$ – solution s neighborhood

$N(s^*)$ – solution s^* neighborhood

$M(s)$ – a proper subset of $N(s)$ which contains all non-tabu or allowed by aspiration solutions from $N(s)$

$f(s)$ – solution s objective function value

TS Algorithm

```
1. Find initial solution $s^0$. $s := s^0, s^* = s^0$.
2. While termination criterion is not satisfied do {
        Select $s := argmin\big(f(s')\big), \forall s' \in M(s)$
        If $f(s) < f(s^*)$ then $s^* = s$
        Declare the move tabu. If some moves' tabu statuses expire in the current
        iteration declare such moves non-tabu.

   }
```

4.2 Unified Tabu Search Algorithm

We use the UTSA by Jean-François Cordeau, Gilbert Laporte and Anne Mercier (2001) as the basis for our metaheuristic algorithms. This metaheuristic has proved to be good on a number of combinatorial problems in accuracy and flexibility.

UTSA takes its beginning from Taburoute algorithm which was built for VRPs with capacity and route length restrictions. The first version of Taburoute was presented in 1991 by Gendreau, Hertz and Laporte. According to Vlček (2005) Taburoute algorithm was very successful in its times and it even provided the best known solutions for many benchmark problem instances in that times. Further, Taburoute led to the development of the UTSA. UTSA is not capable of solving SVPDPMC thus it needs to be reworked.

4.3 Metaheuristics for SVPDPMC

Both of our new metaheuristics includes the most of the TS key concepts:

- Algorithm stopping criteria
- Tabu aspiration
- Continuous diversification
- Strategic oscillation
- Dynamic tabu tenure

Intra-route optimization which was included in the UTSA is omitted.

Besides these concepts new constructed algorithms will include additional features of some other methods that we will explain and talk about in the next section.

4.3.1 Load, Duration and Time Windows Constraints

Load infeasibility is denoted as $q(s)$ and it is the sum of load infeasibilities of all customers. As we are dealing with multicommodities case, so the load infeasibility for a single customer is also a sum of the load infeasibilities of all commodities for this particular customer. If the customer is visited twice then the load infeasibility is calculated twice. Commodity load infeasibility is a difference between the actual load of the vehicle for this commodity and its maximal available capacity also for this commodity. The actual load of the vehicle increases when it picks commodities up at some customer locations and decreases when the vehicle delivers commodities. In our algorithm we assume that we store all commodities in different vehicle compartments – commodities are not mixed. Thus, we have a separate capacity limit for every commodity:

$$\sum_{i=1}^{n}(d_{ih} + p_{ih}) \leq Q_h, 1 \leq h \leq k$$

where n is the number of customers and k is a number of commodities, Q_h is the vehicle maximal load in commodity h, d_{ih} is a customer i delivery demand in commodity h, and p_{ih} is a customer i pickup demand in commodity h.

Duration infeasibility is denoted as $d(s)$. This is the difference between the actual time spent by a vehicle for traveling and servicing all customers and the maximal allowed time limit. Travel duration between two customers can be understood as a cost of traveling between these two customers in our particular case. In other cases it can be named as a length of the route between customers. Customer service time is calculated as a service time for all deliveries plus the time for all pickups for this customer.

To avoid negative values of duration infeasibility we calculate $d(s)$ as $max\{0, t - T\}$, where t is the actually spent time and T is its maximally allowed time limit.

Being able to solve problems with time window constraints was not the aim of this Master thesis. Although we decided to add this possibility as it does not require big changes in either algorithm or its implementation. So, let's denote time windows infeasibility as $w(s)$. For each customer i we then have a time window $[a_i, l_i]$, where a_i is the beginning of a time window and l_i is its end. Vehicles may arrive before the time window starts and then they will have to wait $W_i \coloneqq a_i - A_i$ amount of time, where W_i is a waiting time at location i, A_i is the actual vehicle arrival time at i. We use soft time window constraints without penalty paid for being late. Therefore, we may formulate the total time windows infeasibility equal to:

$$w(s) = \sum_{i=1}^{n} \max\{0, a_i - l_i\}$$

where the notation is the same as above.

We also use the concept of forward time slack described by Cordeau, Laporte and Mercier (2001). This is the time slack on which we can postpone the beginning of the customer service without causing violation of any time windows for other customers.

4.3.2 Neighborhood Structures

The central basic unit of our tabu algorithm is an attribute. Attributes are used to construct solutions, to manipulate these constructed solutions, and to perform moves. They are used in the diversification and aspiration.

The first type of attributes include pairs (i, v) where i is a customer and v is a number of visits to customer i. Attributes form a set $B_1(s) = \{(i, v) | \forall i = \overline{1, n}; v \geq 1; v \in N\}$, where n is a number of all customers for a particular problem instance. Thus a set $B_1(s)$ is a set of all attributes presented in s.

The second type of attributes include triples (i, v, w) where i is some customer, v is a precedent customer to i in the route, and w is a succeeding customer. Analogically we have a set of attributes

$B_2(s) = \{(i, v, w) | \forall i, v, w = \overline{1, n}; v \text{ is a predecessor of } i, w \text{ is a successor of } i\}$.

Tabu search solution s neighborhood is a set of all solutions that are reachable from s by applying a move procedure. Metaheuristics that we develop and describe in this thesis deal with two different move types and hence two different neighborhood structures.

The first type of move changes the number of visits to customers. Since we consider problem instances where each customer can be visited by a vehicle maximum 2 times, this move adds the second visit to the customer if this customer was visited only once and removes the second visit from the route if the customer was visited twice.

The second type of move removes a customer from the route and reinserts it back. As we restrict customers being inserted in the position they were removed from, each time we apply the move we obtain a new solution.

The set of all possible moves of some type for a solution s gives a solution neighborhood. Since two types of moves are available we have one neighborhood with solutions constructed by the move of the type one and another neighborhood with solutions constructed by the move of the type two.

4.3.3 Move Type 1

To make a move of this type means to perform one of the two following operations.

Deleting the 2nd Visit to Customer

The neighborhood is obtained by deleting the second visit to all customers which are visited twice. In terms of attributes it means that we replace an attribute $(i, 2) \in B_1(s)$ of a current solution s with the new attribute $(i, 1) \in B_1(s')$ and that leads to a new solution s'. The replacement is made to maximize $f(s') = \Delta c(s') + \alpha q(s') + \beta d(s') + \gamma w(s')$, where $\Delta c(s')$ is the difference between the cost of s' and the cost of the currently available solution s, α, β, γ – self-adjusted parameters, we will dwell on them later in this thesis. $q(s')$ is a solution s' load infeasibility, $d(s')$ is a solution s' duration infeasibility and $w(s')$ is a solution s' time windows infeasibility.

If a triangle inequality holds for all traveling costs then this procedure of removing the second visit gives solutions with the less cost. However, it may lead to reduction in duration infeasibility and increasing in time windows and / or load infeasibility.

There is one nuance in this procedure that may result in an infinite loop, lost solutions and as a result in a wasted time. To our knowledge this situation was not considered in heuristic implemented by Gjengstø and Vaksvik.

The solution in fact is a sequence of vertices visited by a vehicle. This sequence has duplicated nodes what means that these nodes are visited twice. When the second visit is deleted it is always the last entry of a node that is removed from the sequence. One can imagine the situation that this last entry is located between visits to the same node. In this case deletion of this second visit will result in a self-loop of the neighboring node. It is not allowed as it has no sense in a VRP context. The example is shown in the following figure.

$$0 - 1 - 2 - 3 - \underbrace{4} - \boxed{2} - \underbrace{4} - 1 - 0$$

Figure 7: The route which may lead to the improper solution

Here the second visit to the vertex 2 is a subject to deletion. After the visit is removed one gets a sequence with a self-loop in a vertex 4: 0 – 1 – 2 – 3 – 4 – 4 – 1 – 0. To resolve this loop we need to delete the second visit to vertex 4 as well. Then we get a normal sequence

0 – 1 – 2 – 3 – 4 – 1 – 0. The situation like this actually gives two deletions of second visits for two consequent vertices.

Inserting the 2nd Visit to Customer

For each vertex which is visited once, i.e. for each attribute $(i, v) \in B_1(s)$ where $v < 2$ of the solution s, we try to insert the second visit. It means that we add the attribute $(i, 2) \in B_1(s')$ for the new solution s'. All possible positions where the second visit can be inserted are checked and the position that gives the minimal $f(s') = \Delta c(s') + \alpha q(s') + \beta d(s') + \gamma w(s')$ value is accepted. The notation is the same as in the deletion of the second visit description. As far as we assume that all travel costs satisfy triangle inequality we can affirm that adding the second visit will always give a solution with the higher cost, because $\Delta c(s')$ will always be positive. However, inserting the second visit can give feasible solutions even if there are no feasible Hamiltonian solutions. This especially concerns load infeasibility which is usually decreased after the second visit is inserted. Duration and time windows infeasibility can be increased.

4.3.4 Move Type 2

The move of this type removes a customer from the route and reinserts it back in some other place. The neighborhood for this move is constructed by the following algorithm.

Step 1. Take the first vertex which hasn't been processed with this procedure. Remove it from the route if it will not change the number of visits for other vertices. After removing the vertex from the route restore the path between its predecessor and its successor.
Step 2. Reinsert the removed vertex back into the route in a different position such that $f(s') = \Delta c(s') + \alpha q(s') + \beta d(s') + \gamma w(s')$ will be minimal among all possible insertions. The notation here is the same as for the move type 1.
Step 3. Continue with the step 1 until vertices which have not been processed with this procedure exist.

4.3.5 Building the Initial solution

Initial solution can be build by any constructional heuristic as there are no any additional requirements that this solution should be feasible.

To build initial solution Gjengstrø and Vaksvik, guided by the fact that the initial solution can be infeasible, suggest using Sweep Algorithm. We decided to use the Cheapest Insertion heuristic with the hope that it will give solutions with better initial cost values and that because of this the least cost solutions will be reached earlier.

The main idea of our approach is to insert the customer in such position that the least added insertion cost will be minimal. Concept of added cost is heavily utilized in our algorithm. The added cost is calculated with the following formula.

$$\Delta c = c_{ik} + c_{kj} - c_{ij}, \forall i, j \in V$$

where V is a vertex set in our graph that corresponds to a route, c_{lm} is a cost of traveling from some vertex l to vertex m, where $l, m \in V$. Vertex $k \in V$ is a vertex which insertion is evaluated. Finally, Δc is the added cost of inserting the vertex k between vertices i and j. So, algorithm for construction the initial solution can be formulated in two following steps.

Step 1. Since vertices are represented with Euclidean coordinates, we can sort them in the order of increasing of the angle they make with some arbitrary radius drawn from the depot. In algorithm implementation we use Shellsort algorithm as it gives O(n) best case performance and O(nlog^{2}n) the worst case performance, using the big O notation. In other words we use Shellsort because it is one of the fastest algorithms for sorting comparatively small number of elements.

Step 2. In the beginning the initial solution route is empty. Algorithm iterates through all vertices in a sequence formed on the previous step and inserts them into the initial solution route using the following algorithm:

- Algorithm traverses all vertices starting from the depot and stops when the vertex, which time windows starts later than that of the vertex we want to insert, is found
- Starting from the found position algorithm looks through the rest of the route and inserts a new customer in such position that the appropriate added cost described before is minimal

4.3.6 Stopping Criteria

We decided to choose the original stopping criterion that was used in the original UTSA by Cordeau, Laporte and Mercier, namely – to stop when the pre-specified number of iterations is reached. Additionally we impose the limit on the maximal running time of the algorithm equal to 2 hours. To our personal opinion this is a reasonable time for solving one problem of the size we have. However, in some other cases more time can be needed, for example when the number of customers is higher.

4.3.7 Aspiration Criteria

To prevent situations when attributes that lead to better feasible solutions are under tabu and thus possible better solutions cannot be reached we use aspiration procedure. Here is the example. Let some solution attribute (i, v) be tabu. Hence, (i, v) cannot be selected to perform any move. However, it is known that this attribute leads to a better solution s^*. The load, duration and time windows feasibility constraints of this better solution are satisfied, i.e. $q(s^*) = 0$, $d(s^*) = 0$ and $w(s^*) = 0$ accordingly. So, in perspective we could obtain a good, feasible solution s^* if only (i, v) is not under tabu. In case if we do not have any aspiration mechanism we would definitely loose such solution.

The aspiration procedure checks attributes every time when a move is evaluated. For every attribute (i, v) there exists the aspiration level σ_{iv}. Whenever this aspiration level is reached for some solution s' and s' is feasible, the attribute (i, v) that leads to this solution is declared non-tabu. Each attribute's initial level of aspiration is set to infinity: $\sigma_{iv} \coloneqq \infty$. As far as some attribute is included in a feasible solution s' it's aspiration level is recalculated and set equal to $\sigma_{iv} := \min\{\sigma_{iv}, c(s')\}$, where $c(s')$ is a solution s' cost.

The same is also true for attribute (i, v, w) and its aspiration level σ_{ivw}.

So, attributes can be used in the search process either if they are not tabu or they are allowed by aspiration. The set of such admissible attributes for a solution s is denoted as $M(s) \subseteq N(s)$, where $N(s)$ is a neighborhood of is a neighborhood of s.

4.3.8 Diversification

We decided to use continuous diversification for both of our new metaheuristics. Briefly, it may be described like this. For every attribute from the search space we track the history of its usages in all found solutions. Based on this knowledge we always can get attributes that presented in found solutions more rarely than others. Encouraging the algorithm to use these attributes more often we make the search process to move to the less explored regions of the solution search space.

To diversify the search we use generalized objective function on the step when we need to select the best solution of all possible in a neighborhood. This function for some solution s, which is a result of a move realized by the attribute (i, v), is calculated as follows

$$g(s) = f(s) + p(s)$$

and it is the sum of an auxiliary objective function $f(s)$ and penalty function $p(s)$. Auxiliary objective function will be discussed later. Penalty function is calculated in the following way:

$$p(s) = \zeta c(s)\rho_{iv}/\lambda$$

Here ζ is a multiplier for manual control of a diversification process, $c(s)$ is a cost of the solution s, ρ_{iv} is a frequency of the attribute (i, v) being selected during the search process and λ is a number value of the current iteration. For attribute (i, v, w) penalty function is the similar: $p(s) = \zeta c(s)\rho_{ivw}/\lambda$, where ρ_{ivw} is a frequency of appearing of the attribute (i, v, w) in solutions. $g(s)$ is completely the same.

Assuming that $p(s') = 0$ and hence $g(s') = f(s')$ for the solution s' if $f(s') < f(s), \forall s \neq s',$ we get a generalized objective function $g(s)$ that evaluates each solution and diversifies the search process.

4.3.9 Strategic Oscillation

Strategic oscillation is implemented through the penalized objective function. Let's assume that we have a load violation denoted as $q(s)$, duration violation denoted as $d(s)$, and time windows violation denoted as $w(s)$ for some solution s. The cost of solution is $c(s)$. Total load, duration and time windows violation values are calculated as sums of load, duration and time windows violation values of each customer visit appropriately. If the customer is visited more than once than its total violation comprises violation values from all its visits. So, we can build a function that reflects not only solution cost but also its feasibility.

$$f(s) = c(s) + \alpha q(s) + \beta d(s) + \gamma w(s)$$

α, β, γ are the adjusted parameters that regulate the search process and allow algorithms to accept even non feasible solutions.

In algorithms implementation it is not always convenient to use this function as it is due to the calculation overheads. Thus, we use auxiliary penalized objective function which looks almost the same as $f(s)$ with the only difference that $c(s)$ component is replaced with $\Delta c(s)$. $\Delta c(s)$ is the difference between the cost of the new solution s and the cost of the current solution. So, when $\Delta c(s) < 0$ it means that the cost of the new solution is less than that of the current solution.

α, β, γ parameters are modified in accordance with the information about the search process history. If our algorithms find out that the solution s achieved on the current iteration satisfies the load constraint then current α value is divided by factor $(1 + \delta)$. If it violates the load constraint then its value is multiplied by this factor. δ is calculated on every iteration as a multiplication of the initial δ_0 value and some uniformly distributed random number in the interval [0, 1]. δ_0 is set manually at the beginning of the search process.

The same situation stands with the duration and time windows violation parameters. If the solution satisfies the duration constraint then β is divided by $(1 + \delta)$, otherwise it is multiplied by this value. If the solution satisfies the time windows constraint then γ is divided by $(1 + \delta)$, and is multiplied otherwise.

4.3.10 Tabu Tenure

Currently we use dynamic tabu tenure. Once an attribute is included in a good solution its tabu tenure is updated and set equal to

$$\theta \coloneqq \theta_0 rand(0,1)$$

where θ_0 is the initial pre-set value and $rand(0,1)$ is a random number that is recalculated on each algorithm iteration. The challenge here hides in selecting the initial parameter θ_0 in such a way that attributes are not declared tabu for too long, as it can lead to the lack or absence of non-tabu attributes. Attributes should not be declared for too short as well, as it can lead to cycling in a search process.

4.3.11 Neighborhood Change

We have described two neighborhood constructing procedures:

- The first procedure searches in the solutions space by changing the number of visits for a particular vertex. The neighborhood is obtained by the move of type 1.
- The second procedure removes existing vertex and reinsert it back to the route. The neighborhood is obtained by the move of type 2.

Both neighborhoods are then tried for a solution improvement. Let's denote neighborhoods constructed by these two methods N_1 and N_2 accordingly.
Two metaheuristics we develop in this thesis are based on the following two approaches.

Probabilistic Neighborhood Selection

The first approach deals with the analogy to adaptive large neighborhood search algorithm described by Ropke and Pisinger (2006).
First of all we assign the first neighborhood structure the weight w_1 and the second neighborhood structure – the weight w_2. Initially these weights have equal values $\omega_1 := 1.0$ and $\omega_2 := 1.0$ but during the search process they are adjusted to reflect the superiority of one neighborhood constructing algorithm over another.
Then a roulette wheel selection principle is used to choose the neighborhood that will be used for seeking for the solution improvement. So, for both of the possible neighborhood structures the probability to be selected is calculated by the following formula:

$$p_i = \frac{\omega_i}{\sum_{j=1}^{2} \omega_j}, \qquad i = 1,2.$$

where p_i is the probability of the neighborhood i to be selected, and ω_i is its weight.
Weights w_i should be automatically adjusted on each iteration in accordance with the new knowledge about the current solution process.

After this our metaheuristic will use the neighborhood structure that was selected on the previous step. Selection is made probabilistically with the respect to historical data. Such data represents the information about how often the appropriate neighboring structure improved the solution. The better the structure performs the higher rate it gets and hence the higher possibility to be selected it will have in the future.
Ropke and Pissinger suggested accepting not only heuristics (we use "methods" but not necessarily heuristics) that improve solutions but also heuristics that diversify the search. Therefore, weights of neighborhood construction methods $\omega_i, i \in \{1, 2\}$ will be adjusted in accordance with the following formula:

$$\omega_i = \sum_{j=1}^{3} \sigma_j, \qquad i = 1, 2$$

where

σ_1 is the addition value for the case when selection of a neighborhood structure i has led to the solution improvement;

σ_2 is the addition value for the case when selection of a neighborhood structure i has led to the solution improvement and the solution has never been accepted before;

σ_3 is the addition value for the case when selection of a neighborhood structure i has led to the worse solution but the solution has never been accepted before.

One more algorithm feature is a simulated annealing acceptance rule. Each time we get a new solution s^* we accept it with the probability

$$P(s, s^*) = e^{-\frac{f(s^*)-f(s)}{T}}$$

Where $T > 0$ is a temperature that is decreased on each iteration by some cooling rate parameter $\mu \in]0; 1[: T' = \mu T$. s is a previous solution, f is a solution cost function.
The maximum starting value of T influences the process highly. The number which should be selected depends on a particular problem. To make the search process more diversified we should assign it a big number, for example 100,000.

It is essential to give a chance to the algorithm to deteriorate current solution what may perhaps lead to the best solutions in future.

One can notice that probabilistic neighborhood selection approach needs lots of memory to keep all obtained solutions and needs processor time to check if the solution has already been accepted before or not. Ropke and Pisinger suggested using hashing with a hashtable for storing such solutions. We investigated this issue and came to several conclusions. First of all, we decided not to save the whole solutions as they are, for we would need a lot of memory to store them. We therefore store a fake value for each found solution and assume that the chosen hash function is safe enough to avoid collisions. We took solution cost as a fake value since it gives slightly better protection from collisions than just simply “any” value. So, when we want to know if a solution has ever been found before, we do not only calculate its hash function and get the cost from the hash set, but also check if this cost from the hash set is equal to the cost of the current solution. If this cost exists in the hash set and it is not equal then we make a conclusion that a collision has occurred. For hashing we decided to use Professor Daniel J. Bernstein function, since it was reported to be one of the most efficient hash functions. Below one can find two variants of the same hash function.

$$h(i) = 33h(i-1) + v_i,$$
$$h(i) = \big(h(i-1) \ll 5 \ + h(i-1)\big) + \ v_i$$

where $h(i)$ is a hash’s i-component for the customer $v_i, i = 1,2,\dots,n$, and n – is the problem size, $\ll$ is a left binary shifting operator.

Finally, we added dynamic memory allocation to our metaheuristic. Static memory allocation that was used in the UTSA was not acceptable due to the increased requirements to memory usage. Summarizing, we get a working method capable of saving the found solutions efficiently and performing their search in a constant time. Hash function is calculated in O(n) time though, where n is a number of customers in a route.

Variable Neighborhood Selection

The second approach uses Variable neighborhood search (VNS). It is described by Mladonović and Hansen (1997).

In general case as it is stated by Mladonović and Hansen we have a set of neighborhood structures with some cardinality k among which we iterate to get the next solution improvement. For our method we have only two neighborhood structures, thus $k = 2$ and neighborhood structures are N_1 and N_2. Hence, the VNS algorithm can be written as follows:

Step 1. Let's assume that we have some solution s whether initial or received on the previous steps of the algorithm.

Step 2. Then select neighborhood structure $N_1(s)$ and try to improve the solution s and get a new solution s'. If an old solution s can be improved and better solution s' can be found, set $s := s'$ and go the next algorithm iteration, otherwise select the neighborhood structure $N_2(s)$ and try it for improvements on the next iteration. If no improvements can be found in $N_2(s)$, switch to $N_1(s)$.

Step3. Use the number of iterations as a stopping criterion.

The following figure shows the process of the variable neighborhood change search process.

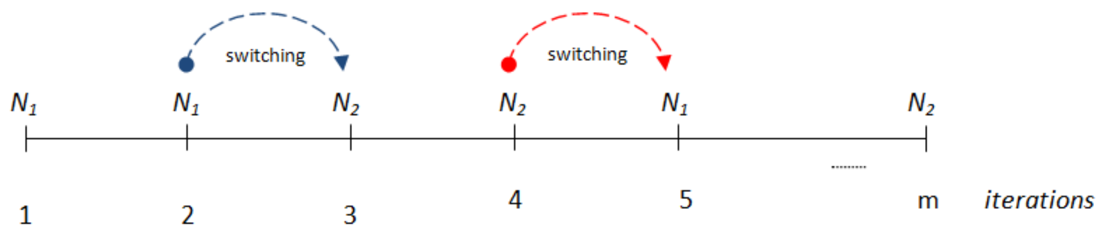

Figure 8: The example of changing the neighborhood constructing procedures

4.3.12 Metaheuristics Template

Now let's describe our two metaheuristics step by step. To do this we need the following notation. Both metaheuristics can be described by the same template. Metaheuristic that uses probabilistic neighborhood selection is run when $\xi = 1$ and metaheuristic that uses variable neighborhood selection – when $\xi = 2$.

Notation

(i, v) – attribute, where i is a customer and v is a number of visits to the customer i

$B(s)$ – set of attributes for the solution s

$c(s)$ – solution s cost

$q(s)$ – vehicle load violation during the route for given solution s

$M(s)$ – set of non-tabu attributes and also those allowed by aspiration

$s, \tilde{s}$ – solutions

s_0 – initial solution

s^* – best solution found

α – penalty parameter for load violation

β – penalty parameter for duration violation

γ – penalty parameter for time window violation

δ – factor for updating α, β, γ

ζ – factor used to adjust the intensity of the diversification

η – total number of iteration to be performed

θ – tabu duration

λ – iteration counter

ξ – neighborhood construction algorithm, currently $\xi \in \{1,2\}$

ρ^1_{iv} – number of times attribute (i, v) has been added to the solution applying algorithm $\xi = 1$

σ^1_{iv}– aspiration level of attribute (i, v) for algorithm $\xi = 1$

τ^1_{iv}– last iteration for which attribute (i, v) is declared tabu after applying algorithm $\xi = 1$

ρ^2_{ivw} – number of times attribute (i, v, w) has been added to the solution applying algorithm $\xi = 2$

σ^2_{ivw}– aspiration level of attribute (i, v, w) for algorithm $\xi = 2$

τ^2_{ivw}– last iteration for which attribute (i, v, w) is declared tabu after applying algorithm $\xi = 2$

φ – parameter used for intra-route optimization

T – time limit for running algorithm

$N(s, \xi)$ – solution s neighborhood that is build using the neighborhood construction algorithm ξ

p_i – probability to select $i = 1, 2$ neighborhood, $\sum_{i \in \{1,2\}} p_i = 1$

ω_i – the weight of the neighborhood i

T – temperature

μ – cooling rate

Before the algorithm starts parameters η, δ_0, ζ_o, θ_0, φ should be set manually. Good choice of these values can make a difference and result in better solutions being found faster.

```
1. Obtain initial solution $s_o$ applying cheapest insertion heuristic described above.
2. If $s_0$ is feasible, then $s^* \coloneqq s_0$.
3. $s \coloneqq s_0,\ \alpha \coloneqq 1,\ \beta \coloneqq 1,\ \gamma \coloneqq 1$
4. For every attribute $(i, v)$ do
    a. $\rho_{iv}^1 \coloneqq 0,\ \tau_{iv}^1 \coloneqq 0$
    b. If $(i, v) \in B(s)$, then $\sigma_{iv}^1 \coloneqq c(s)$, otherwise $\sigma_{iv}^1 \coloneqq \infty$
5. For every attribute $(i, v, w)$ do
    a. $\rho_{ivw}^2 \coloneqq 0,\ \tau_{ivw}^2 \coloneqq 0$
    b. $\sigma_{iv}^2 \coloneqq \infty$
6. $\lambda \coloneqq 1,\ \xi \coloneqq 1$
7. While $\lambda \neq \eta$ or $T$ is reached
    a. $\delta \coloneqq \delta_0 rand(0,1)$
       $\zeta \coloneqq \zeta_o rand(0,1)$
       $\theta \coloneqq \theta_0 rand(0,1)$
    b. $N(s, \xi) \coloneqq \emptyset,\ M(s) \coloneqq \emptyset$
    c. Create a set of solutions $N(s, \xi)$ applying a $\xi$ move algorithm if possible:
        i. If $\xi = 1$, then
            • For each $(i, v') \notin B(s)$ find $s'$ by replacing $(i, v) \in B(s)$ with $(i, v')$ and calculate $f(s') \coloneqq \Delta c(s') + \alpha q(s') + \beta d(s') + \gamma w(s')$
            • $N(s, \xi) \coloneqq N(s, \xi) \cup s'$
           Else
            • For each $(i, v, w) \in B(s)$ find $s'$ by replacing $(i, v, w)$ with $(i, v', w') \in B(s) \cup \overline{B(s)}$, where $(v' \neq v \mid w' \neq w)$ and calculate $f(s') \coloneqq \Delta c(s') + \alpha q(s') + \beta d(s') + \gamma w(s')$
```

```
d. If $N(s,\xi) \neq \emptyset$, then
    i. For each $s' \in N(s,\xi)$ do
        • If $\xi = 1$, then
                For $(i,v) \in B(s')\backslash B(s)$ such that $\tau_{iv}^1 < \lambda$ or ($s'$ is feasible and $c(s') < \sigma_{iv}^1$), set $M(s) \coloneqq M(s) \cup s'$
          Else
                For $(i,v,w) \in B(s')\backslash B(s)$ such that $\tau_{ivw}^2 < \lambda$ or ($s'$ is feasible and $c(s') < \sigma_{ivw}^2$), set $M(s) \coloneqq M(s) \cup s'$
    ii. For each $s' \in M(s)$ do
        • If $f(s') \geq f(s)$, then $g(s') \coloneqq \begin{cases} f(s') + \frac{\zeta c(s)\rho_{iv}^1}{\lambda}, if\ \xi = 1 \\ f(s') + \frac{\zeta c(s)\rho_{ivw}^2}{\lambda}, otherwise \end{cases}$
          Else $g(s') \coloneqq f(s')$
    iii. Find $s' \in M(s)$ for which $g(s')$ is minimal
    iv. If $\xi = 1$, then
        • For each $(i,v) \in B(s')\backslash B(s)$, $\rho_{iv}^1 \coloneqq \rho_{iv}^1 + 1$, $\tau_{iv}^1 \coloneqq \lambda + \theta$
        Else
        • For each $(i,v,w) \in B(s')$, $\rho_{ivw}^2 \coloneqq \rho_{ivw}^2 + 1$, $\tau_{ivw}^2 \coloneqq \lambda + \theta$
    v. If $s'$ is feasible, then
        • If $\xi = 1$, then for each $(i,v) \in B(s')$ $\sigma_{iv}^1 \coloneqq \min\{\sigma_{iv}^1, c(s')\}$
          Else for each $(i,v,w) \in B(s')$ $\sigma_{ivw}^2 \coloneqq \min\{\sigma_{ivw}^2, c(s')\}$
        • $\alpha \coloneqq \begin{cases} \frac{\alpha}{1+\delta}, if\ q(s') = 0 \\ \alpha(1+\delta), otherwise \end{cases}$
        • $\beta \coloneqq \begin{cases} \frac{\beta}{1+\delta}, if\ d(s') = 0 \\ \beta(1+\delta), otherwise \end{cases}$
        • $\gamma \coloneqq \begin{cases} \frac{\gamma}{1+\delta}, if\ w(s') = 0 \\ \gamma(1+\delta), otherwise \end{cases}$
        • $P(s, s') = e^{-\frac{f(s') - f(s)}{T}}$
        • Accept $s \coloneqq s'$ with probability $P(s, s')$
```

```
            • If ($s$ is accepted and $c(s') < c(s*)$), then $s* := s'$
        vi. Update weights $\omega_i, i = 1,2$ in accordance with the current $\xi$
            value
    Else
            • If we use VNS, then $\xi := \xi + 1\ (mod\ 2)$ else
              select $\xi$ probabilistically: $\xi = 1$ has the probability $p_1 = \frac{\omega_1}{\sum_{j=1}^{2} \omega_j}$ to be selected and $\xi = 2$ has the probability $p_2 = \frac{\omega_2}{\sum_{j=1}^{2} \omega_j}$.
e. $\lambda := \lambda + 1$
f. $T := \mu T$
```

5 Computational Experiments

5.1 *Environment*

Our metaheuristics were programmed in C. For IDE we chose Sun NetBeans 6.5 as it provides nice features and is free of charge. To compile and to debug the application we use a port of the GNU Compile Collection and GNU Binutils - MinGW version 5.1.4. We also use MSYS 1.0, for it has a working make-command component.

Tests were run on a HP Compaq 6820s with Intel Core2Duo 1.60 GHz processor, 1GB RAM with 32 bit Windows Vista Home Premium.

We took the same test instances as described in Gjengstrø and Vaksvik (2008). Below we have a comparison between our metaheuristics and heuristic by Gjengstrø and Vaksvik. We compare these methods in solutions optimality, solutions convergence to the optimal value and time needed to complete all algorithm iterations. We also compare solutions' shapes.

Gjengstrø and Vaksvik's heuristic best found results were taken from their work whereas running time was measured on the same computer where our metaheuristics were tested. To compile and run Gjengstrø and Vaksvik's heuristic we used Pelles C 2.90.8.

Randomness in the construction procedure of the initial solution results in different initial routes. This leads to the distinct final solutions. Running a program several times we may get different results. Therefore, the testing was performed in the following way: all instances were run once and then, those instances on which algorithms performed badly were run several times more. For the instances that were run multiple times we chose the best solutions among all obtained ones.

5.2 *Initial Parameters*

Maximal number of iterations η is taken equal to 100,000 for all test instances. Other initial parameters are as follows:

$\delta_0 \coloneqq 1.2,$

$\zeta_o \coloneqq 1.2,$

$\theta_0 \coloneqq 10 \lg (10n),$

$\varphi \coloneqq 5.$

For probabilistic neighborhood selection the initial parameters are:

$\sigma_1 \coloneqq 0.05,$

$\sigma_2 \coloneqq 1.00,$

$\sigma_3 \coloneqq 0.05,$

$T \coloneqq 100000,$

$\mu \coloneqq 0.999999.$

Service time for each commodity is taken equal to 1 time unit per 1 commodity unit. Thus, to serve a customer a vehicle spends one time unit per each unit of delivered commodity and one time unit per each unit of picked up commodity.

5.3 Description

First of all we should say that our heuristics is not expected to surpass heuristic by Gjengstrø and Vaksvik on all test instances. However, we expect our heuristic to obtain the better results very often.

Let's recall that neighborhoods in our heuristic can be obtained by changing the number of visits to customers (that is when customers can be visited once or twice) and by changing the order of customers in a route. When the vehicle load is 100% of its maximal capacity most of the Hamiltonian-shaped solutions become load infeasible. We will show that at the same time the most of the optimal solutions become non-Hamiltonian-shaped. Therefore, more thorough search is needed among solutions that have at least one customer which is visited twice – such solutions are provided by the neighborhood which is constructed by changing the number of visits. The second neighborhood which is constructed by changing the sequence of customers is not supposed to influence the search process highly. Taking that into account we can say that in a 100% vehicle load case our algorithms will behave approximately the same as the heuristic of Gjengstrø and Vaksvik. When the vehicle load decreases, more solutions with single visited customers become load feasible. Among such solutions we may encounter solutions with the better cost than any of feasible non-Hamiltonian-shaped. So, we expect our heuristic to give better results on solution instances with less than 100% of the vehicle load. We also want to find such vehicle load level when the most of found solutions are Hamiltonian-shaped.

In test instances the vehicle maximum capacity of some commodity is calculated as the maximum from two sums: the sum of all demands of this commodity and the sum of all pickups of this commodity.

We test our metaheuristics on 40 small and 13 large instances that were taken from Gjengstrø and Vaksvik (2008). These instances were generated by Gjengstrø and Vaksvik as follows.

The first batch includes instances of two types. Instances of the first type that have "b20" in their names were generated from CVRP instances. For each customer i its delivery demand d_i and pickup demand p_i were calculated as follows:

$$d_i := d_i^o, q_i := \begin{cases} \lfloor 0.8 q_i^o \rfloor, \text{if } i \text{ is even} \\ \lfloor 1.2 q_i^o \rfloor, \text{otherwise} \end{cases}, i = 1,2,\dots,n$$

where d_i^o is the original demand for CVRP problem instance, n is the problem size.

The instances of the second type which do not have any suffix in their names have a dependency between customers' demands and their geographic locations. Customers' coordinates and delivery demands are the same as for the first type instances of the same name. The 30% of customers which are closest to the depot got a pickup demand equal to 120% of their delivery demand. The next 30% of customers got a pickup demand equal to 80% of their delivery demand. The rest of customers got a pickup demand calculated by the formula for the first type instances.

The second batch of instances was generated from instances at http://www.fernuni-hagen.de/WINF/touren/probleme/. Each generated test instance has the name consisting of the number of nodes in it and the name of the original instance. The number of customers was selected randomly from 5 to 25 and then a random original instance was reduced to this size. The pickup demands for the first commodity were calculated in the same was as for the first type of instances from the first bunch.

Large problem instances were generated from the CVRP instances taken from VRPLIB. These are 13 instances that have from 41 to 262 customers together with the depot. The CVRP instances were downloaded from http://www.or.deis.unibo.it/research_pages/ORinstances/VRPLIB/VRPLIB.html.

Delivery demand d_{i1} of customer i and pickup demand p_{i1} of i of commodity 1 were generated as follows:

$$d_{i1} := d_i^o, q_{i1} := \begin{cases} \lfloor 0.8 q_i^o \rfloor, \text{if } i \text{ is even} \\ \lfloor 1.2 q_i^o \rfloor, \text{otherwise} \end{cases}, i = 1,2,\dots,n$$

where d_i^o is the original demand for CVRP problem instance, n is the problem size.
Delivery and pickup demands of the second commodity are the reversed delivery and pickup demands of the first commodity: $d_{i2} \coloneqq d_{n+1-i,1}, q_{i2} \coloneqq q_{n+1-i,2}$, where n is the number of customers and $i = \overline{1,n}$.
Delivery and pickup demands of the third commodity are the arithmetic mean of the corresponding delivery and pickup demands of the first and the second commodities of the same customer multiplied by the random number from the interval [0.6, 1.6].
Small and large instances are presented in two flavors: 2-commodities instances (name starts from 2c) and 3-commodities instances (name starts from 3c). After this the name of an instance includes the number of vertices in it. The rest of the name uniquely identifies the instance.

5.4 Testing on Small Instances

To compare the results of three metaheuristic algorithms we present them in tables. With G&V we mark the results given by Gjengstrø and Vaksvik's algorithm, with T-VNS – the results of our tabu search based metaheuristic with variable neighborhood change approach, with T-Probabilistic – the results of our tabu search based metaheuristic with probabilistic neighborhood change approach, with CPLEX – the results from CPLEX.
The following tables show the gap between the named three algorithms' solutions and CPLEX solution. All these gaps are presented in percents. Gaps mean deviation from the CPLEX solution. Negative deviation means that CPLEX solution was improved, 0% means that the algorithm obtained exactly the same result as CPLEX and positive deviation means that obtained solution is worse than that of the CPLEX. Thus, the less the deviation is, the better. Problem instances on which heuristics outperform or obtained the same solutions are marked in bold.
We present 2-commodities case divided into 3 separate tables for convenience.

Instance	G&V (in %)	T-VNS (in %)	T-Probabilistic (in %)
2c_1c_6_c1_6_3	**0.00**	**0.00**	**0.00**
2c_1c_6_c110_3	**0.00**	**0.00**	**0.00**
2c_1c_7_c1_2_2	**0.00**	**0.00**	**0.00**
2c_1c_8_c1_4_3	**0.00**	**0.00**	**0.00**
2c_1c_8_c110_5	**0.00**	**0.00**	**0.00**
2c_1c_9_c1_2_7	**0.00**	**0.00**	**0.00**
2c_1c_11_c1_2_9	**0.00**	**0.00**	**0.00**
2c_1c_11_c110_7	**0.00**	**0.00**	**0.00**
2c_1c_12_c110_1	**0.00**	**0.00**	**0.00**
2c_1c_13_c1_4_4	2.00	1.00	**0.00**
2c_1c_13_c1_6_1	**0.00**	**0.00**	**0.00**
2c_1c_14_c110_9	**0.00**	**0.00**	**0.00**
2c_1c_15_c1_2_5	4.00	1.00	**-2.00**
2c_1c_15_c1_4_6	3.00	**0.00**	**0.00**
2c_1c_16_c110_8	**0.00**	**0.00**	**0.00**
2c_1c_19_c1_4_2	7.00	2.00	**-1.00**
2c_1c_19_c1_210	4.00	**-2.00**	**-2.00**
2c_1c_19_c110_4	**-2.00**	**-2.00**	**-2.00**
2c_1c_20_c1_4_7	4.00	**-1.00**	**-1.00**
2c_1c_20_c1_6_4	**-1.00**	**-1.00**	**-1.00**
2c_1c_21_c1_2_3	**-11.00**	**-11.00**	**-14.00**
2c_1c_22_c110_6	3.00	**0.00**	**0.00**
2c_1c_23_c1_2_8	**-11.00**	**-11.00**	**-11.00**
2c_1c_23_c1_4_9	8.00	**0.00**	**-1.00**
2c_005	**0.00**	**0.00**	**0.00**
2c_006	**0.00**	**0.00**	**0.00**
2c_016	**0.00**	**0.00**	**0.00**
2c_016b20	**0.00**	**0.00**	**0.00**
2c_021	**0.00**	**0.00**	**0.00**
2c_021b20	**0.00**	**0.00**	**0.00**
2c_022	**0.00**	**0.00**	**0.00**
2c_022b20	**0.00**	**0.00**	**0.00**
2c_023	**0.00**	**0.00**	**0.00**
2c_023b20	**-2.00**	**-2.00**	**-2.00**
2c_026	4.00	3.00	**0.00**
2c_026b20	4.00	3.00	1.00
2c_030	**-3.00**	**-5.00**	**-6.00**
2c_030b20	**-15.00**	**-17.00**	**-17.00**
2c_031	1.00	**0.00**	**0.00**
2c_031b20	1.00	**0.00**	**0.00**

Table 6a: 2 commodities case, 90% vehicle load

Instance	G&V (in %)	T-VNS (in %)	T-Probabilistic (in %)
2c_1c_6_c1_6_3	**0.00**	**0.00**	**0.00**
2c_1c_6_c110_3	**0.00**	**0.00**	**0.00**
2c_1c_7_c1_2_2	**0.00**	**0.00**	**0.00**
2c_1c_8_c1_4_3	**0.00**	**0.00**	**0.00**
2c_1c_8_c110_5	**0.00**	**0.00**	**0.00**
2c_1c_9_c1_2_7	**0.00**	**0.00**	**0.00**
2c_1c_11_c1_2_9	**0.00**	**-1.00**	**-1.00**
2c_1c_11_c110_7	**0.00**	**0.00**	**0.00**
2c_1c_12_c110_1	**0.00**	**0.00**	**0.00**
2c_1c_13_c1_4_4	2.00	1.00	**0.00**
2c_1c_13_c1_6_1	**0.00**	**0.00**	**0.00**
2c_1c_14_c110_9	**0.00**	**0.00**	**0.00**
2c_1c_15_c1_2_5	5.00	5.00	2.00
2c_1c_15_c1_4_6	**-1.00**	**-1.00**	**-1.00**
2c_1c_16_c110_8	**0.00**	**0.00**	**0.00**
2c_1c_19_c1_4_2	8.00	2.00	**0.00**
2c_1c_19_c1_210	4.00	**-1.00**	**-1.00**
2c_1c_19_c110_4	**0.00**	**0.00**	**0.00**
2c_1c_20_c1_4_7	4.00	**-1.00**	**-1.00**
2c_1c_20_c1_6_4	**-10.00**	**-10.00**	**-10.00**
2c_1c_21_c1_2_3	**-9.00**	**-9.00**	**-9.00**
2c_1c_22_c110_6	3.00	**0.00**	**0.00**
2c_1c_23_c1_2_8	**-18.00**	**-18.00**	**-20.00**
2c_1c_23_c1_4_9	**0.00**	**-8.00**	**-8.00**
2c_005	**0.00**	**0.00**	**0.00**
2c_006	**0.00**	**0.00**	**0.00**
2c_016	**0.00**	**0.00**	**0.00**
2c_016b20	**0.00**	**0.00**	**0.00**
2c_021	**0.00**	**0.00**	**0.00**
2c_021b20	**0.00**	**0.00**	**0.00**
2c_022	**0.00**	**0.00**	**0.00**
2c_022b20	**0.00**	**0.00**	**0.00**
2c_023	**-2.00**	**-2.00**	**-2.00**
2c_023b20	-	-	-
2c_026	2.00	1.00	**0.00**
2c_026b20	4.00	3.00	**0.00**
2c_030	**-9.00**	**-11.00**	**-11.00**
2c_030b20	**-14.00**	**-16.00**	**-16.00**
2c_031	1.00	**0.00**	**0.00**
2c_031b20	1.00	**0.00**	**0.00**

Table 6b: 2 commodities case, 95% vehicle load

Instance	**G&V (in %)**	**T-VNS (in %)**	**T-Probabilistic (in %)**
2c_1c_6_c1_6_3	**0.00**	**0.00**	**0.00**
2c_1c_6_c110_3	**0.00**	**0.00**	**0.00**
2c_1c_7_c1_2_2	**0.00**	**0.00**	**0.00**
2c_1c_8_c1_4_3	**0.00**	**0.00**	**0.00**
2c_1c_8_c110_5	**0.00**	**0.00**	**0.00**
2c_1c_9_c1_2_7	**0.00**	**0.00**	**0.00**
2c_1c_11_c1_2_9	**-1.00**	**-1.00**	**-1.00**
2c_1c_11_c110_7	**0.00**	**0.00**	**0.00**
2c_1c_12_c110_1	**0.00**	**0.00**	**0.00**
2c_1c_13_c1_4_4	**0.00**	**0.00**	**0.00**
2c_1c_13_c1_6_1	**0.00**	**0.00**	**0.00**
2c_1c_14_c110_9	**0.00**	**0.00**	**0.00**
2c_1c_15_c1_2_5	**-6.00**	**-6.00**	**-6.00**
2c_1c_15_c1_4_6	**-2.00**	**-2.00**	**-2.00**
2c_1c_16_c110_8	**-1.00**	**-1.00**	**-1.00**
2c_1c_19_c1_4_2	**-7.00**	**-7.00**	**-7.00**
2c_1c_19_c1_210	**-24.00**	**-24.00**	**-24.00**
2c_1c_19_c110_4	**-1.00**	**-1.00**	**-1.00**
2c_1c_20_c1_4_7	**0.00**	**-1.00**	**-1.00**
2c_1c_20_c1_6_4	**-6.00**	**-6.00**	**-6.00**
2c_1c_21_c1_2_3	**-8.00**	**-8.00**	**-8.00**
2c_1c_22_c110_6	2.00	**-3.00**	2.00
2c_1c_23_c1_2_8	**-31.00**	**-31.00**	**-31.00**
2c_1c_23_c1_4_9	**-29.00**	**-29.00**	**-29.00**
2c_005	**0.00**	**0.00**	**0.00**
2c_006	**0.00**	**0.00**	**0.00**
2c_016	**0.00**	**0.00**	**0.00**
2c_016b20	**0.00**	**0.00**	**0.00**
2c_021	**0.00**	**0.00**	**0.00**
2c_021b20	**0.00**	**0.00**	**0.00**
2c_022	**-3.00**	**-3.00**	**-3.00**
2c_022b20	**0.00**	**0.00**	**0.00**
2c_023	**0.00**	**-1.00**	**0.00**
2c_023b20	**-1.00**	**-1.00**	**-1.00**
2c_026	6.00	4.00	**0.00**
2c_026b20	**0.00**	**0.00**	**0.00**
2c_030	**-4.00**	**-6.00**	**-6.00**
2c_030b20	**-16.00**	**-16.00**	**-18.00**
2c_031	1.00	**0.00**	**0.00**
2c_031b20	1.00	**-1.00**	**-1.00**

Table 6c: 2 commodities case, 100% vehicle load

For problem instance 2c_023b20 with 95% vehicle load there is no gap listed in a table, for CPLEX was not able to find any feasible solution in 1 hour according to Gjengstrø and Vaksvik.

0% deviation from CPLEX solution may indicate that we found a global optimum as well as that we are in a local optimum. CPLEX results were taken from the work of Gjengstrø and Vaksvik where the running time was limited to one hour. Therefore, negative deviation happens when CPLEX does not succeed in finding the optimal solution in one hour time limit, or when it produces an error after running for some period of time. In both cases the best found solution was taken without any guarantee in its global optimality.

When a vehicle load is equal to 90%, in 5 cases for Variable neighborhood search heuristic and in 1 case for Probabilistic neighborhood search heuristic we have worse results than for CPLEX. When vehicle capacity is equal to 95% we have worse results in 5 cases for Variable neighborhood approach and in 1 case for Probabilistic approach. When vehicle capacity is fully utilized, i.e. its load is 100%, worse results are obtained 1 time both for Variable neighborhood selection approach and for Probabilistic neighborhood selection approach. So, Probabilistic approach slightly outperforms Variable neighborhood approach in what concerns times when the algorithm outputs are worse than those of CPLEX. As we see, the more the vehicle load is, the less number of problems are worse. It occurs so because on more constrained instances, when vehicle capacity is fully utilized, CPLEX needs more time to find good results and, hence, more often the time limit it has is insufficient.

Heuristic by Gejngstrø and Vaksvik performed worse on 12 test instances with 90% load, on 10 test instances with 95% load, and on 4 instances with 100% vehicle load. So, both our algorithms outperform G&V heuristic in the number of solutions that are not worse than those of CPLEX.

Now let's look at the average values. G&V algorithm was worse on average on 3.75% in 90% load case, on 3.4% in 95% load case, and on 2.5% in 100% load.

Variable neighborhood search algorithm performed worse on average on 2% in 90% load case, on 2.4% in 95% load case, and on 4.0% in 100% load case. 4.0% is a comparatively big number but we should not forget that this algorithm performed worse only on one problem instance where it had 4.0% gap. On the same problem instance G&V algorithm performed much worse - it showed 6.0% gap, and only small gaps on other problem instances decreased its average gap.

Probabilistic neighborhood selection algorithm was worse on average on 1.0% in 90% load case, on 2.0% in 95% load case, and on 2.0% in 100% load case.

G&V heuristic outperforms our two algorithms on 100% vehicle load on average only because of the bigger number of problem instances where the deviation from CPLEX results is positive and comparatively small. On all instances where the gap is positive our algorithms often produce solutions of the less cost.

The next tables represent results for instances with 3 commodities.

Instance	**G&V (in %)**	**T-VNS (in %)**	**T-Probabilistic (in %)**
3c_1c_6_c1_6_3	**0.00**	**0.00**	**0.00**
3c_1c_6_c110_3	**0.00**	**0.00**	**0.00**
3c_1c_7_c1_2_2	**0.00**	**0.00**	**0.00**
3c_1c_8_c1_4_3	**0.00**	**0.00**	**0.00**
3c_1c_8_c110_5	**0.00**	**0.00**	**0.00**
3c_1c_9_c1_2_7	**0.00**	**0.00**	**0.00**
3c_1c_11_c1_2_9	**0.00**	**0.00**	**0.00**
3c_1c_11_c110_7	**0.00**	**0.00**	**0.00**
3c_1c_12_c110_1	**0.00**	**0.00**	**0.00**
3c_1c_13_c1_4_4	2.00	**0.00**	**0.00**
3c_1c_13_c1_6_1	**0.00**	**0.00**	**0.00**
3c_1c_14_c110_9	**0.00**	**0.00**	**0.00**
3c_1c_15_c1_2_5	**-3.00**	**-6.00**	**-9.00**
3c_1c_15_c1_4_6	**-12.00**	**-14.00**	**-14.00**
3c_1c_16_c110_8	5.00	**0.00**	**0.00**
3c_1c_19_c1_4_2	**-2.00**	**-9.00**	**-9.00**
3c_1c_19_c1_210	**-8.00**	**-13.00**	**-13.00**
3c_1c_19_c110_4	**0.00**	**0.00**	**0.00**
3c_1c_20_c1_4_7	**0.00**	**-5.00**	**-5.00**
3c_1c_20_c1_6_4	**-3.00**	**-4.00**	**-4.00**
3c_1c_21_c1_2_3	**-9.00**	**-9.00**	**-9.00**
3c_1c_22_c110_6	3.00	**0.00**	**0.00**
3c_1c_23_c1_2_8	**-10.00**	**-10.00**	**-10.00**
3c_1c_23_c1_4_9	6.00	4.00	**-2.00**
3c_005	**0.00**	**0.00**	**0.00**
3c_006	**0.00**	**0.00**	**0.00**
3c_016	**0.00**	**0.00**	**0.00**
3c_016b20	**0.00**	**0.00**	**0.00**
3c_021	**0.00**	**0.00**	**0.00**
3c_021b20	**0.00**	**0.00**	**0.00**
3c_022	**-1.00**	**-1.00**	**-1.00**
3c_022b20	**0.00**	**0.00**	**0.00**
3c_023	**0.00**	**0.00**	**0.00**
3c_023b20	**-2.00**	**-2.00**	**-2.00**
3c_026	4.00	4.00	4.00
3c_026b20	4.00	2.00	1.00
3c_030	**-12.00**	**-17.00**	**-19.00**
3c_030b20	**-8.00**	**-10.00**	**-11.00**
3c_031	1.00	**0.00**	**0.00**
3c_031b20	1.00	**0.00**	**0.00**

Table 7a: 3 commodities case, 90% vehicle load

Instance	G&V (in %)	T-VNS (in %)	T-Probabilistic (in %)
3c_1c_6_c1_6_3	**0.00**	**0.00**	**0.00**
3c_1c_6_c110_3	**0.00**	**0.00**	**0.00**
3c_1c_7_c1_2_2	**0.00**	**0.00**	**0.00**
3c_1c_8_c1_4_3	**0.00**	**0.00**	**0.00**
3c_1c_8_c110_5	**0.00**	**0.00**	**0.00**
3c_1c_9_c1_2_7	**0.00**	**0.00**	**0.00**
3c_1c_11_c1_2_9	1.00	1.00	**-1.00**
3c_1c_11_c110_7	**0.00**	**0.00**	**0.00**
3c_1c_12_c110_1	**0.00**	**0.00**	**0.00**
3c_1c_13_c1_4_4	2.00	**0.00**	**0.00**
3c_1c_13_c1_6_1	**0.00**	**0.00**	**0.00**
3c_1c_14_c110_9	**0.00**	**0.00**	**0.00**
3c_1c_15_c1_2_5	**-2.00**	**-6.00**	**-8.00**
3c_1c_15_c1_4_6	**-6.00**	**-6.00**	**-6.00**
3c_1c_16_c110_8	5.00	5.00	**0.00**
3c_1c_19_c1_4_2	2.00	**-4.00**	**-6.00**
3c_1c_19_c1_210	**-20.00**	**-21.00**	**-20.00**
3c_1c_19_c110_4	**-2.00**	**-2.00**	**-2.00**
3c_1c_20_c1_4_7	**-4.00**	**-8.00**	**-8.00**
3c_1c_20_c1_6_4	**-5.00**	**-6.00**	**-6.00**
3c_1c_21_c1_2_3	**-13.00**	**-13.00**	**-13.00**
3c_1c_22_c110_6	**0.00**	**0.00**	**0.00**
3c_1c_23_c1_2_8	**-15.00**	**-16.00**	**-16.00**
3c_1c_23_c1_4_9	**-1.00**	**-4.00**	**-3.00**
3c_005	**0.00**	**0.00**	**0.00**
3c_006	**0.00**	**0.00**	**0.00**
3c_016	1.00	**0.00**	**0.00**
3c_016b20	**0.00**	**0.00**	**0.00**
3c_021	**0.00**	**0.00**	**0.00**
3c_021b20	**0.00**	**0.00**	**0.00**
3c_022	**0.00**	**0.00**	**0.00**
3c_022b20	**-1.00**	**-1.00**	**-1.00**
3c_023	**-2.00**	**-2.00**	**-2.00**
3c_023b20	**0.00**	**0.00**	**0.00**
3c_026	2.00	**-1.00**	**0.00**
3c_026b20	2.00	2.00	2.00
3c_030	**-5.00**	**-8.00**	**-15.00**
3c_030b20	**-15.00**	**-16.00**	**-17.00**
3c_031	1.00	1.00	**0.00**
3c_031b20	1.00	**0.00**	**0.00**

Table 7b: 3 commodities case, 95% vehicle load

Instance	G&V (in %)	T-VNS (in %)	T-Probabilistic (in %)
3c_1c_6_c1_6_3	**0.00**	**0.00**	**0.00**
3c_1c_6_c110_3	**0.00**	**0.00**	**0.00**
3c_1c_7_c1_2_2	**0.00**	**0.00**	**0.00**
3c_1c_8_c1_4_3	**0.00**	**0.00**	**0.00**
3c_1c_8_c110_5	**0.00**	**0.00**	**0.00**
3c_1c_9_c1_2_7	**0.00**	**0.00**	**0.00**
3c_1c_11_c1_2_9	**-1.00**	**-1.00**	**-1.00**
3c_1c_11_c110_7	**-5.00**	**0.00**	**0.00**
3c_1c_12_c110_1	**-10.00**	**-3.00**	**0.00**
3c_1c_13_c1_4_4	**0.00**	**0.00**	**0.00**
3c_1c_13_c1_6_1	**0.00**	**0.00**	**0.00**
3c_1c_14_c110_9	**0.00**	**0.00**	**0.00**
3c_1c_15_c1_2_5	**-14.00**	**-14.00**	**-14.00**
3c_1c_15_c1_4_6	**-4.00**	**-4.00**	**-4.00**
3c_1c_16_c110_8	**-4.00**	**-4.00**	**-4.00**
3c_1c_19_c1_4_2	**-5.00**	**-5.00**	**-5.00**
3c_1c_19_c1_210	**-15.00**	**-15.00**	**-15.00**
3c_1c_19_c110_4	**-2.00**	**-2.00**	**-2.00**
3c_1c_20_c1_4_7	**-3.00**	**-4.00**	**-4.00**
3c_1c_20_c1_6_4	**-6.00**	**-6.00**	**-6.00**
3c_1c_21_c1_2_3	**-24.00**	**-24.00**	**-24.00**
3c_1c_22_c110_6	2.00	2.00	**0.00**
3c_1c_23_c1_2_8	**-31.00**	**-31.00**	**-31.00**
3c_1c_23_c1_4_9	**-22.00**	**-21.00**	**-21.00**
3c_005	**0.00**	**0.00**	**0.00**
3c_006	**0.00**	**0.00**	**0.00**
3c_016	**0.00**	**0.00**	**0.00**
3c_016b20	**0.00**	**0.00**	**0.00**
3c_021	**0.00**	**0.00**	**0.00**
3c_021b20	**0.00**	**0.00**	**0.00**
3c_022	**-5.00**	**-5.00**	**-5.00**
3c_022b20	**-3.00**	**-3.00**	**-3.00**
3c_023	**0.00**	**0.00**	**0.00**
3c_023b20	**-3.00**	**-3.00**	**-3.00**
3c_026	**-7.00**	**-7.00**	**-7.00**
3c_026b20	**0.00**	**0.00**	**0.00**
3c_030	**-5.00**	**-5.00**	**-5.00**
3c_030b20	**-20.00**	**-21.00**	**-23.00**
3c_031	1.00	**0.00**	**0.00**
3c_031b20	-	**-**	**-**

Table 7c: 3 commodities case, 100% vehicle load

In 3c_031b20 instance CPLEX was not able to find any solution in a time limit of one hour, thus we have "-" in cells for all three algorithms.

On problem instances with 90% vehicle load Variable neighborhood showed quite good results: it was worse than CPLEX on 3 problem instances; Probabilistic neighborhood was even better: it was worse only on 2 instances.

Variable neighborhood gave worse results in 4 problem instances when vehicle load was increased to 95% and Probabilistic selection appeared to be worse on 1 problem instance.

Finally, on 100% vehicle load, Variable neighborhood was worse on 1 instance and Probabilistic neighborhood gave the same or better results on all problem instances.

G&V heuristic was then CPLEX on 8 instances with 90% vehicle load, on 9 instances with 95% load, and on 2 instances with 100% vehicle load. As with 2 commodities case it is seen that our algorithms outperformed G&V heuristic.

If we take the average deviations from the CPLEX results for problem instances with 3 commodities we will get the following. G&V algorithm is worse than CPLEX on 3.25% whereas T-VNS is worse on 3.33% and T-Probabilistic is worse on 2.5% for 90% vehicle load instances. On 95% vehicle load problem instances G&V algorithm is worse than CPLEX on 1.89%, T-VNS on 2.25%, and T-Probabilistic algorithms on 2.0%. On 100% vehicle load instances G&V has 1.5%, T-VNS has 2.0% gap, and T-Probabilistic algorithm is always at least as good as CPLEX.

The next tables represent the average values for three algorithms for small problem instances where the number of customers is less than 31, with 2 and 3 commodities. The same as before: the less the gap is, the better.

Average values 90% load			
	G&V	**T-VNS**	**T-Probabilistic**
2 commodities	0.00%	-1.05%	-1.48%
3 commodities	-1.10%	-2.25%	-2.58%

Table 8a: Average gaps from CPLEX, 90% load

Average values 95% load			
	G&V	**T-VNS**	**T-Probabilistic**
2 commodities	-0.74%	-1.69%	-2.00%
3 commodities	-1.85%	-2.63%	-3.05%

Table 8b: Average gaps from CPLEX, 95% load

Average values 100% load			
	G&V	T-VNS	T-Probabilistic
2 commodities	-3.25%	-3.60%	-3.60%
3 commodities	-4.77%	-4.51%	-4.54%

Table 8c: Average gaps from CPLEX, 100% load

The numbers represent the mean values on all solutions costs for all problem instances we tested our algorithms on. Thus, we can see how G&V, T-VNS, and T-Probavilistic algorithms perform on average on all available problem instances. The dependence on the number of commodities in a problem and vehicle load can also be tracked from these tables.

So, we see that T-Probabilistic approach outperforms T-VNS on average in all cases. Changing the number of commodities also influences the mean results: increasing the number of commodities in problem instances lead to better results on average in comparison with CPLEX and it is true for all three algorithms. One can see that T-VNS and T-Probability heuristics outperforms on average heuristic by Gejngstrø and Vaksvik. However, on problem instances with full capacity utilization and 3 commodities (Table 8c) the deviations of T-VNS and T-Probabilistic heuristics are greater. It happens not because our two algorithms perform worse than CPLEX on more instances than G&V algorithm but only because G&V heuristic was getting solutions with much smaller costs than CPLEX more often.

In general, all three algorithms drastically improve CPLEX results on problem instances with full capacity utilization. Heuristic by Gejngstrø and Vaksvik is especially good with 100% load cases but in 95% and 90% load cases T-VNS and T-Probabilistic algorithms on average show better results.

Table 9 shows how many solutions found by G&V, T-VNS and T-Probabilistic algorithms are not worse than the best known solutions found by CPLEX.

2 commodities			
Load	G&V	T-VNS	T-Probabilistic
90	70%	87.5%	97.5%
95	75%	87.5%	97.5%
100	90%	97.5%	97.5%
3 commodities			
Load	G&V	T-VNS	T-Probabilistic
90	80%	92.5%	95%
95	77.5%	90%	97.5%
100	95%	97.5%	100%

Table 9: Percentage of instances where algorithms found better or at least the same solutions as CPLEX

From one side, one may see that T-VNS and T-Probabilistic heuristics produce more solutions solved to the optimality than G&V heuristic. From the other side, one may also notice that all three algorithms do slightly worse than CPLEX, except T-Probavilistic for 3 commodities case with 100% vehicle load. Although, we must admit that the numbers in the table include solutions that appeared to be better than solutions that CPLEX has been able to find. Tabu algorithms have quite close results to CPLEX but in general they spend less time.

To compare the number of instances on which T-VNS and T-Probabilistic heuristics outperformed G&V heuristic let's look at the table 10.

2 commodities				
	Better than G&V (in %)		Worse than G&V (in %)	
Load	T-VNS	T-Probabilistic	T-VNS	T-Probabilistic
90	35	37.5	0	0
95	32.5	37.5	0	0
100	17.5	15	0	0
3 commodities				
	Better than G&V (in %)		Worse than G&V (in %)	
Load	T-VNS	T-Probabilistic	T-VNS	T-Probabilistic
90	37.5	37.5	0	0
95	32.5	37.5	0	0
100	7.5	10	7.5	7.5

Table 10: Percentage of instances where metaheuristics obtained better/worse results than G&V

T-VNS and T-Probabilistic heuristics obtained worse results than G&V heuristic only on set of problem instances with 100% vehicle load and 3 commodities. On other problem instances we see that our algorithms performed better. Adding additional commodities, solving 3 commodities' instances rather than 2 commodities', seems to make more difficult for both T-VNS and T-Probabilistic to find better solutions. Not significantly difficult though. Load factor also influences the behavior in comparison with G&V: the fully the load capacity is utilized the rarer we get the less cost solutions. It can be explained: the lower the load is, the more Hamiltonian-shaped solutions become feasible, and, thus, the more profitable becomes to use neighborhood that deals with only changing the order of visits but not the number of visits (please, see the T-VNS and T-Probabilistic algorithms description section for details about neighborhood construction approaches). The higher the load is, the less feasible solutions have Hamiltonian shape, and hence, changing the order of visits in the algorithm does not have much sense which makes solutions closer to G&V outputs. T-Probabilistic heuristic then obtains better results as it can adapt to the situations when changing the order of customers in a route becomes unprofitable, - it simply starts to use changing the number of visits approach more often.

2 commodities		
Load	**T-VNS (in %)**	**T-Probabilistic (in %)**
90	17.5	7.5
95	17.5	5
100	62.5	62.5
3 commodities		
Load	**T-VNS (in %)**	**T-Probabilistic (in %)**
90	15	5
95	17.5	7.5
100	67.5	70

Table 11: Percentage of found non-Hamiltonian solutions

Table 11 demonstrates how the number of commodities and vehicle load level influence solution shapes. Increasing the load from 90% to 95% as well as increasing the number of commodities from 2 to 3 does not influence the number of non-Hamiltonian solutions highly. Nevertheless, increasing the vehicle load from 95% to 100% gives the high growth of the number of non-Hamiltonian-shaped solutions on all problem instances. So, the main tendency is that with increasing the vehicle load the number of non-Hamiltonian solutions increases, too. It is reasonable, since when the load is high, one will more often encounter feasible non-Hamiltonian solutions when some customers should be visited more than once not to violate load constraints. In such solutions one will observe pickup and delivery being made in separate customers' visits. The more the load is increased the more Hamiltonian-shaped solutions become infeasible, and, thus, the more non-Hamiltonian-shaped solutions become optimal. This is exactly what we see in the Table 11.

However, as it has been already said before, if the algorithm obtains some optimal non-Hamiltonian solution, it does not mean that there are no feasible Hamiltonian-shaped solutions in the search space at all. It means that some can exist but be not optimal.

What also should be mentioned is that all solutions that have the second customers visits, have only one customer which is visited twice. For such customer delivery is made in the first visit and pickup – on the second.

In the following figures one can see reports on the time needed to run 100000 iterations of G&V, T-VNS, and T-Probabilistic heuristics with different number of commodities and of different vehicle load.

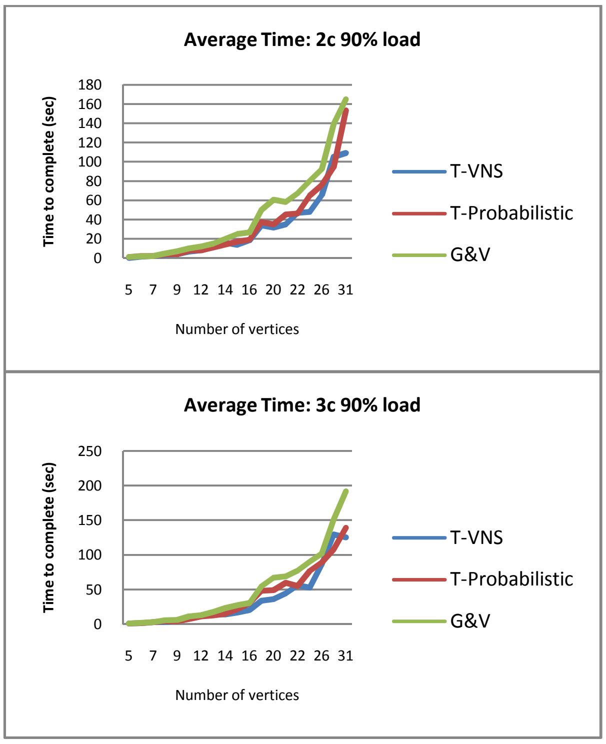

Figure 6a: Time needed to complete 100,000 iterations, 2/3 commodities, 90% vehicle load

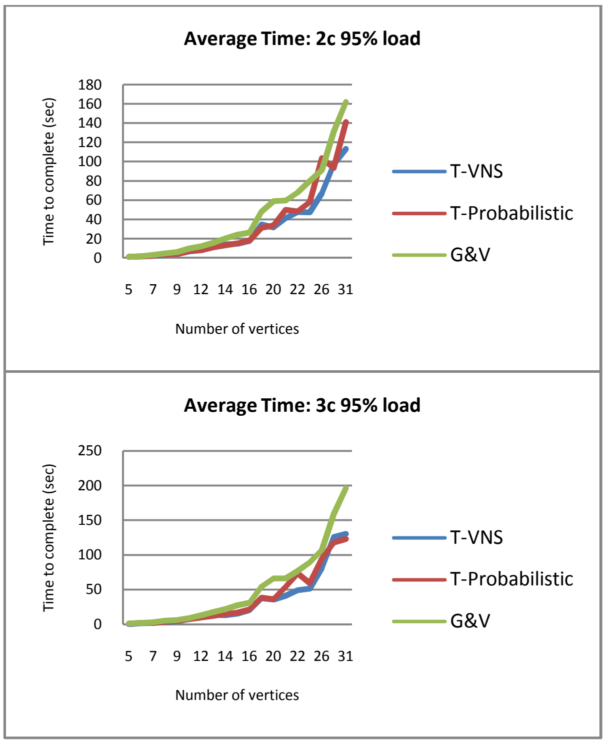

Figure 6b: Time needed to complete 100,000 iterations, 2/3 commodities, 95% vehicle load

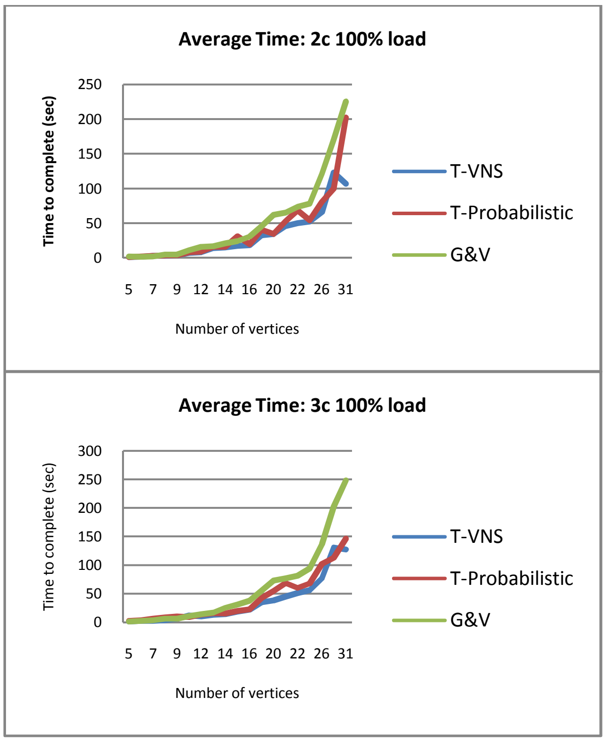

Figure 6c: Time needed to complete 100,000 iterations, 2/3 commodities, 100% vehicle load

The curves represent the average running times which are the average times one needs to solve problem instances with the same number of customers.

Both T-VNS and T-Probabilistic heuristics need less time to perform 100,000 tabu search iterations, hence, in applications we can increase the number of iterations which can lead to better solutions. The bigger the number of vertices in the problem, the bigger is the difference between running times. The fastest algorithm on average is T-VNS but, as we have seen before, this algorithm is not the best one when it comes to solutions' optimality. T-Probabilistic algorithm is just almost as fast as T-VNS but produces better solutions.

5.5 Testing on Large Instances

In this section we test algorithms on a larger problem instances: such instances that have from 41 to 262 vertices.

Test results are listed in tables. As previously, we have problem instances with 2 and 3 commodities with 90%, 95%, and 100% vehicle load.

The following tables represent the results from G&V, T-VNS and T-Probabilistic algorithms. Since, CPLEX solutions were not presented by Gjengstrø and Vaksvik in their work, we can only make a comparison with the results of their heuristic.

As previously: 0 means that our algorithms found the same solution as Gjengstrø and Vaksvik's heuristic. Positive gaps mean that G&V outperformed our metaheuristics, and negative – that our metaheuristics obtained better solutions. Problem instances where metaheuristics outperformed G&V heuristic or obtained the same results are marked in bold.

Instance	**T-VNS (in %)**	**T-Probabilistic (in %)**
2c_041b20	**-1.24**	**-1.24**
2c_045b20	**-0.56**	**-1.6**
2c_048b20	**-0.51**	**-0.51**
2c_051b20	**-2.62**	**-1.73**
2c_072b20	**0**	**0**
2c_076b20	**-1.92**	**-3.23**
2c_101b20	**-0.02**	**-0.88**
2c_111b20	**-2.38**	**-4.98**
2c_121b20	**-8.26**	**-1.67**
2c_135b20	**-10.06**	**-2.51**
2c_151b20	**-4.36**	**-2.54**
2c_200b20	**-0.62**	**-0.73**
2c_262b20	**-2.76**	**-1.78**
Average	**-2.72**	**-1.8**

Table 12a: 2 commodities case, 90% load

Instance	**T-VNS (in %)**	**T-Probabilistic (in %)**
2c_041b20	**-1.56**	**-1.24**
2c_045b20	**-2.01**	**-2.49**
2c_048b20	**-0.58**	**-0.51**
2c_051b20	**-1.46**	**-0.77**
2c_072b20	0.06	**0**
2c_076b20	**-1.98**	**-1.79**
2c_101b20	**-0.34**	**-0.11**
2c_111b20	**-4.98**	**-4.98**
2c_121b20	**-1.52**	**-7.16**
2c_135b20	**-2.67**	**-4.19**
2c_151b20	**-3.35**	**-4.37**
2c_200b20	**-3.03**	**-4.36**
2c_262b20	**-0.18**	**-1.57**
Average	**-1.82**	**-2.58**

Table 12b: 2 commodities case, 95% load

Instance	T-VNS (in %)	T-Probabilistic (in %)
2c_041b20	**0**	**-0.21**
2c_045b20	1.13	0.41
2c_048b20	**-0.53**	**-0.53**
2c_051b20	**-0.9**	**-1.17**
2c_072b20	**-3.45**	**-3.04**
2c_076b20	**-3.4**	**-3.28**
2c_101b20	**-0.8**	0.82
2c_111b20	**-3.6**	**-3.64**
2c_121b20	**-1.24**	5.34
2c_135b20	**-9.22**	**-8.41**
2c_151b20	**-0.39**	**-0.98**
2c_200b20	**-1.03**	**-0.13**
2c_262b20	**-2.6**	**-2.44**
Average	**-2**	**-1.33**

Table 12c: 2 commodities case, 100% load

The following tables represent test results for instances with 3 commodities.

Instance	T-VNS (in %)	T-Probabilistic (in %)
3c_041b20	**-1.24**	**-1.56**
3c_045b20	**-0.56**	**-1.6**
3c_048b20	**-0.84**	**-0.58**
3c_051b20	**-2.62**	**-2.62**
3c_072b20	1.2	**0**
3c_076b20	**-3.75**	**-1.68**
3c_101b20	**-0.63**	**-0.05**
3c_111b20	**-1.62**	**-4.98**
3c_121b20	**-7.43**	**-8.95**
3c_135b20	**-11.42**	**-4.19**
3c_151b20	**-2.47**	**-4.68**
3c_200b20	**-4.39**	**-1.21**
3c_262b20	**-1.36**	**-1.28**
Average	**-2.86**	**-2.57**

Table 13a: 3 commodities case, 90% load

Instance	T-VNS (in %)	T-Probabilistic (in %)
3c_041b20	**-1.56**	**-1.24**
3c_045b20	**-2.49**	**-2.01**
3c_048b20	**-0.49**	**0**
3c_051b20	**-1.01**	**-1.55**
3c_072b20	0.06	0.06
3c_076b20	**-1.68**	**-2.59**
3c_101b20	**-1.47**	**-0.41**
3c_111b20	**-2.82**	**-2.56**
3c_121b20	**-8.09**	**-8.79**
3c_135b20	**-4.23**	**-5.57**
3c_151b20	**-2.26**	**-3.33**
3c_200b20	**-2.62**	**-3.31**
3c_262b20	**-0.8**	**-0.09**
Average	**-2.27**	**-2.41**

Table 13b: 3 commodities case, 95% load

Instance	T-VNS (in %)	T-Probabilistic (in %)
3c_041b20	**-0.29**	**-0.56**
3c_045b20	6.12	6.12
3c_048b20	**-2.05**	**-0.53**
3c_051b20	**-2.89**	**-0.21**
3c_072b20	**-1.35**	**-0.06**
3c_076b20	**-1.84**	**-3.04**
3c_101b20	0.1	0.53
3c_111b20	**-1.27**	**-0.97**
3c_121b20	**-7.51**	**-7.58**
3c_135b20	**-6**	**-0.46**
3c_151b20	1.15	0.29
3c_200b20	1.96	0.51
3c_262b20	**-2.7**	**-3.27**
Average	**-1.27**	**-0.71**

Table 13c: 3 commodities case, 100% load

As one may see, G&V tabu search was able to find solutions for all problem instances as well as our two algorithms – this is a significant feature of heuristics, to be able to find solutions even when exact methods fail.

In case of 2 commodities we obtained the following results.

With 90% of the vehicle load T-VNS and T-Probabilistic were both worse than G&V on 0 problem instances and better on 12. Changing the vehicle load to 95% does not change the situation significantly: T-VNS was worse on 1 test and T-Probabilistic – on 0. Both T-

VNS and T-Probabilistic obtained better results on 12 instances – the same as on 90% load. Further increasing of the load up to 100% influences the results: T-VNS becomes worse on 1 test whereas T-Probabilistic – on 3, T-VNS is better on 11 instances and T-Probabilistic – on 10.

In case of 3 commodities:

On 90% load tests T-VNS is worse in 1 case, T-Probabilistic is worse in 0 cases. Both T-VNS and T-Probabilistic are better on 12 instances. 95% load as in the case of 2 commodities does not change the situation significantly: T-VNS and T-Probabilistic are worse on 1 problem instance, and better – on 12 and 11 instances appropriately. 100% load results are slightly different: T-VNS and T-Probabilistic are worse on 4 tests, and better on 9.

What we see here is that T-VNS and T-Probabilistic heuristics are much better in comparison to G&V on problem instances where vehicle capacity is high. The same conclusion was also made for small problem instances.

Now, let's figure out which algorithm on average is better in comparison to G&V. One may see that for tests with 90% and 100% load, 2 and 3 commodities T-VNS's average gap value is less than T-Probabilistic's. Only on tests with 95%, 2 and 3 commodities T-Probabilistic has the smaller gap. Hence, T-VNS on average shows better results and obtain better solutions in comparison to G&V algorithm than T-Probabilistic. Although, the difference in average gaps is not very high, so we think that situation may change: running T-Probabilistic algorithm more times and selecting the best found solutions may give us another average values. Moreover, we have only 13 large test instances which can be not enough to make a very thorough and accurate analysis.

The following tables depict on how much T-VNS and T-Probabilistic algorithms are worse and better on average and on how many tests these algorithms are worse and better than G&V (numbers are rounded to 2 digits after the full stop).

2 commodities				
	Better than G&V (in %)		Worse than G&V (in %)	
Load	**T-VNS**	**T-Probabilistic**	**T-VNS**	**T-Probabilistic**
90	3.00	2.17	0.00	0.00
95	2.00	2.75	0.00	0.00
100	2.45	2.30	1.00	2.00
3 commodities				
	Better than G&V (in %)		Worse than G&V (in %)	
Load	**T-VNS**	**T-Probabilistic**	**T-VNS**	**T-Probabilistic**
90	3.17	2.92	1.00	0.00
95	2.42	2.91	0.00	0.00
100	2.89	1.89	2.25	2.00

Table 14: On how much on average algorithms are better/worse than G&V

2 commodities				
	Better than G&V (in %)		Worse than G&V (in %)	
Load	**T-VNS**	**T-Probabilistic**	**T-VNS**	**T-Probabilistic**
90	92.31	92.31	0.00	0.00
95	92.31	92.31	7.69	0.00
100	84.62	76.92	7.69	23.08
3 commodities				
	Better than G&V (in %)		Worse than G&V (in %)	
Load	**T-VNS**	**T-Probabilistic**	**T-VNS**	**T-Probabilistic**
90	92.31	92.31	7.69	0.00
95	92.31	84.62	7.69	7.69
100	69.23	69.23	30.77	30.77

Table 15: Percentage of instances where T-VNS and T-Probabilistic metaheuristics obtain better/worse results than G&V

On how much algorithms are better/worse on average are calculated as a mean of all values where algorithms are better/worse. So, 3.17% for T-VNS 90% load, 3 commodities means that T-VNS mean gap among all tests where T-VNS is better than G&V is 3.17%.
One may see that T-Probabilistic results are closer to G&V results: average gaps where it is better as well as worse are smaller than those of T-VNS. Thus, T-VNS results are more widely dispersed. Table 15 demonstrates that increasing the vehicle load from 95% to 100% impacts the number of solutions where our heuristics are better. So, if the vehicle capacity is fully utilized, T-VNS and T-Probabilistic are better on the smaller number of

test instances and are worse on the bigger. What we want to admit is that T-VNS and T-Probabilistic heuristics are better on much bigger number of test instances than they are in case when test instances are small.

Next table show the percentage of non-Hamiltonian solutions.

2 commodities		
Load	**T-VNS (in %)**	**T-Probabilistic (in %)**
90	38.46	38.46
95	23.08	38.46
100	92.31	84.62
3 commodities		
Load	**T-VNS (in %)**	**T-Probabilistic (in %)**
90	23.08	15.38
95	46.15	30.77
100	92.31	100

Table 16: Percentage of found non-Hamiltonian solutions

For large instances we have bigger percentage of non-Hamiltonian solutions among all found solutions than for small instances. Nevertheless, as it has been stated before, after the vehicle load becomes greater than 95% we observe significant increase of non-Hamiltonian-shaped solutions. For 3 commodities case with 100% load it reaches even 100% if T-Probabilistic is used. Comparing these results with the results for small test instances one may notice that the larger the problem size is, the bigger is the number of feasible solutions that have non-Hamiltonian shapes. It may happen because with the increasing of the size of the problem the search space also increases. Hence, to find good solutions algorithms need to put more efforts and look through the bigger number of intermediate solutions. So, by increasing the number of iterations and time limit we can improve results and get more Hamiltonian solutions.

Another observation is that for large problem instances we started to obtain solutions where 2 and even more customers are visited twice. For small problem instances only 1 customer at maximum is visited twice. The next table shows how many instances from 13 tested ones have visits from 0 to 6.

2 commodities						
	90% load		95% load		100% load	
Visits	**T-VNS**	**T-Probabilistic**	**T-VNS**	**T-Probabilistic**	**T-VNS**	**T-Probabilistic**
0	8	8	10	8	1	2
1	4	2	1	4	6	2
2	0	1	1	0	4	3
3	0	1	0	1	2	1
4	1	1	0	0	0	0
5	0	0	1	0	0	0
6	0	0	0	0	0	0
3 commodities						
	90% load		**95% load**		**100% load**	
Visits	**T-VNS**	**T-Probabilistic**	**T-VNS**	**T-Probabilistic**	**T-VNS**	**T-Probabilistic**
0	10	11	7	9	1	0
1	2	0	3	3	7	9
2	0	2	3	0	1	2
3	1	0	0	1	2	1
4	0	0	0	0	0	1
5	0	0	0	0	0	0
6	0	0	0	0	2	0

Table 17: The number of instances that have solutions with 0-6 visits to customers.

From the table above we see that when the vehicle load is increased from 95% to 100% the number of problems that have 2 and more visits drastically increases. When the vehicle load is increased from 90% to 95% we have small increase in the number of best found solutions that have 2 and more visits to customers.

The following charts show how much time algorithms need to run 100,000 iterations. The time limit as it has been already said before is 7200 seconds or 2 hours.

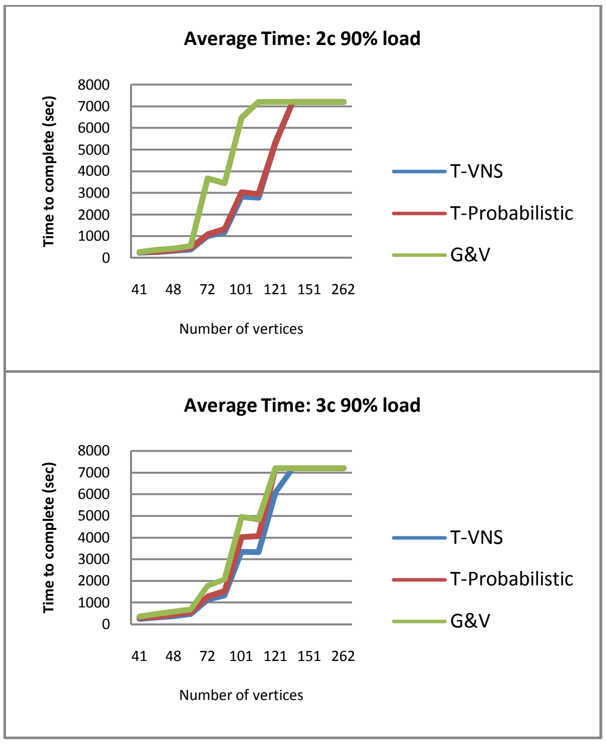

Figure 7a: Time needed to complete 100,000 iterations, 2/3 commodities, 90% vehicle load

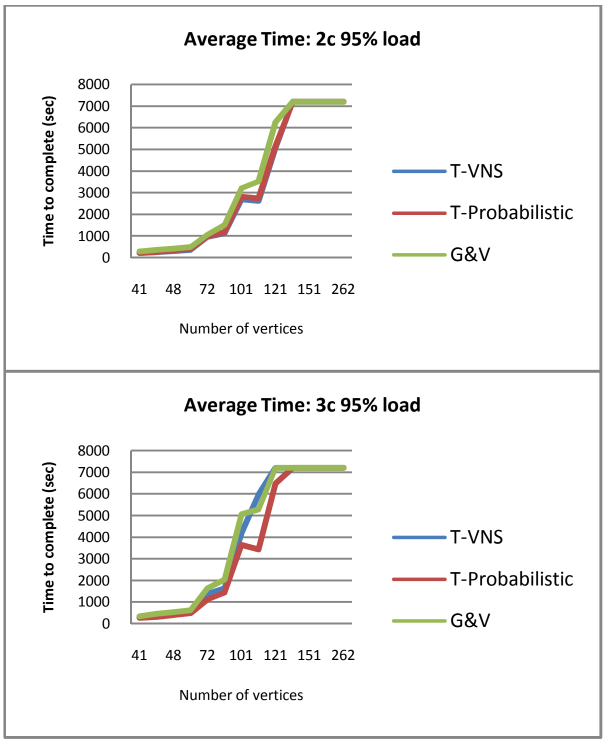

Figure 7b: Time needed to complete 100,000 iterations, 2/3 commodities, 95% vehicle load

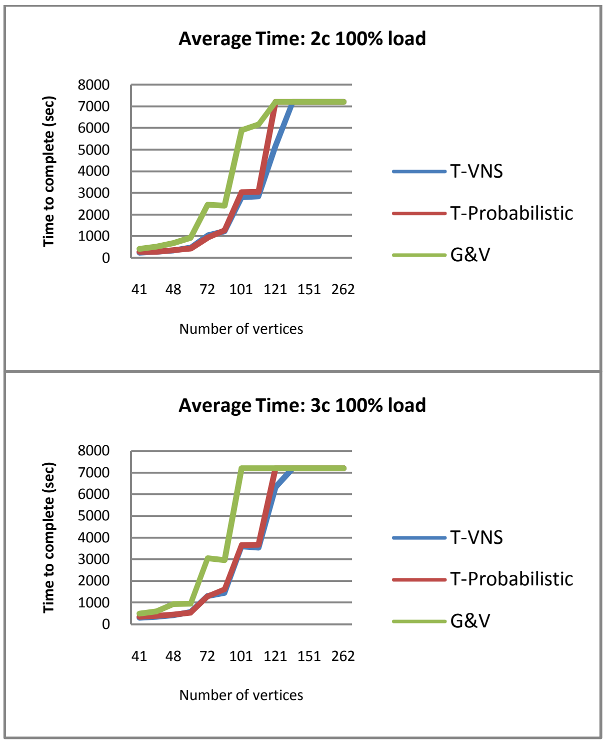

Figure 7c: Time needed to complete 100,000 iterations, 2/3 commodities, 100% vehicle load

As we see from charts, both T-VNS and T-Probabilistic metaheuristics need less time than G&V heuristic to complete 100,000 iterations. Since one needs less time to accomplish the search process, one can perform greater number of iterations in a fixed amount of time and by this improve the results. T-VNS and T-Probabilistic need almost the same time.

6 Conclusions

In this thesis we developed, implemented and tested two new metaheuristic algorithms for solving SVPDPMC. Additionally these metaheuristics can also solve SVPDPMC with Time Windows.

We showed that solution that is optimal in cost is not necessarily Hamiltonian-shaped. Thus, algorithms which can find routes where customers are visited several times along the route may give better costs than have routes with customers visited only once. Probability neighborhood selection and Variable neighborhood selection metaheuristics developed in this thesis allow finding such routes.

Presented metaheuristics outperform previously known heuristic in speed and solutions quality. Metaheuristic with probabilistic neighborhood selection is more technologically advanced than metaheuristic with variable neighborhood selection. The first uses Tabu search metaheuristic, Probabilistic neighborhood selection and Simulated annealing, whereas the latter uses only Variable neighborhood selection with Tabu search.

In general both metaheuristics needed less time and obtained better solutions than heuristic by Gjengstrø and Vaksvik. The time needed for two metaheuristics to complete all tabu search iterations grows slowly with the increasing of the number of customers than the time of Gjengstrø and Vaksvik's algorithm. It allows solving problem instances of the bigger size in the less time.

Variable neighborhood selection and Probabilistic neighborhood selection tabu search based algorithms produce better solutions on more problem instances when the vehicle load is not fully utilized. On problem instances with the load below 95% both algorithms obtain much better solutions than Gjengstrø and Vaksvik's heuristic. Decreasing the load from 95% to 90% does not influence solutions quality highly: we obtain solutions that are only slightly better. On instances with 100% vehicle load both metaheuristics outperform Gjengstrø and Vaksvik's heuristic as well. We noticed that below 95% load level most of found solutions are Hamiltonian-shaped. So, if the vehicle load is less than 95%, then most of the best-known solutions will mainly contain customers who are visited only once, whereas when the vehicle load increases, we observe a drastic increase in the number of best-known solutions with two visits to customers. As it has been mentioned before, this situation is quite obvious, since more Hamiltonian-shaped solutions become load infeasible with the higher level of the vehicle load. Hence, double visits to customers are needed more often.

Increasing the number of commodities in tests did not significantly influence algorithms behavior and solutions.

On large problem instances the percentage of instances where our metaheuristics are better than heuristic by Gjengstrø and Vaksvik is greater than on small problem instances. The number of obtained feasible solutions that have non-Hamiltonian shapes is also bigger.

On small instances all non-Hamiltonian solutions has at most one customer visited twice, whereas on large problem instances the number of customers visited twice increases and reaches 6 in maximum. On large instances Metaheuristic with variable neighborhood selection obtains solutions with the bigger number of customers that are visited twice.

Probabilistic neighborhood selection algorithm in general performed better than variable neighborhood selection. It produced better results more often and of the less costs. Time needed to complete the search was almost the same and the tiny difference can be neglected.

7 Further Research

Due to its nature, Bernstein hash function may lead to collisions when dealing with long solution routes. This may hamper our algorithm to give the right answer to the question if the solution exists in a hash set or not. We performed a small testing to find out how many times this happens. The result was encouraging – there were only one collision on the instance with 261 customers. This is the problem instance with the maximal number of customers among all available problem instances. The bigger the number of customers is - the longer is the route that covers all these customers. The longer the route is - the probability of collisions is higher. Therefore, we may expect that on smaller instances we will get no collisions at all.

Nevertheless, hash function can possibly give more collisions in some other cases. We should also admit, that even if a collision occurs it does not mean that the solution is rejected and is not considered, it only means that the neighborhood will get the wrong addition to its weight. If there are not many collisions then it should not impact solutions quality highly. There are two ways to avoid improper algorithm functioning caused by collisions: to use better hashing algorithm or to utilize an associative array which will store lists of values with the same hashes instead of values by themselves. The first approach is always challenging, the latter is very common and easier but results in O(n) time when looking for a value in such array. n here is the number of algorithm iterations, since on each iteration we obtain one solution that is then saved in the array.

One more issue is the deletion of the second visit to a customer. We defined a solution route as a sequence of customers which starts from the depot and ends with the depot. If a customer is visited twice in some solution it means that it appears twice in the route sequence that represents this solution. To remove the second visit to the customer our algorithm had always to remove the second occurrence of this customer in the solution sequence. This behavior was inherited from Gjengstrø and Vaksvik's algorithm. Despite it works perfectly well it may be a subject for improvement, as solutions which can be obtained by removing the first customer occurrence may be of the less cost.

Another research direction is allowing customers to be visited more than twice. It can probably lead to solution routes of the less cost but this issue is quite doubtful.

Parameters which we used for testing were tuned on the number of selected instances. However, it is very possible that the better combination of their values which leads to much better problem solutions exists.

Some of the problem instances were run several times and the best found solutions for them were tracked. One can try to run algorithms several times on all instances. It can result in better solutions and in better average gaps for both metaheuristics.

References

- Baranov, Valeriĭ Ivanovich; Stechkin, Boris Sergeevich. (1995) *Extremal Combinatorial Problems and Their Applications.* Springer, 38-42.
- Barbeglia, Gerardo; Cordeau, Jean-François; Gribkovskaia, Irina; Laporte, Gilbert. (2007) *Static Pickup and Delivery Problems: A Classification Scheme and Survey.*
- Ceria, S. (1998) *Parallel algorithms for discrete optimization.* Computational Optimization Research Center Columbia University joint with Gabor Pataki. http://www-unix.mcs.anl.gov/metaneos/talks/ceria-html/
- Chinneck, John W. (2007) *Feasibility and Infeasibility in Optimization: Algorithms and Computational Methods.* Springer, 23-25.
- Coja-Oghlan, A.; Krumke,S.O.; Nierhoff, T. (2003) *A Heuristics for the Stacker Crane Problem on trees which is almost surely exact.* Journal of Algorithms 61: 1-19.
- Cordeau, J.-F.; Laporte, G.; Mercier, A. (2001) *A Unified Tabu Search Heuristic for Vehicle Routing Problems with Time Windows.* Journal of the Operational Research Society 52: 928-936.
- Crevier, Benoit; Cordeau, Jean-François; Laporte, Gilbert. (2007) *The multi-depot vehicle routing problem with inter-depot routes*. European Journal of Operational Research 176: 756–773.
- Du, Dingzhu; Pardalos, Panos M. (1998) *Handbook of combinatorial optimization.* Springer.
- Erdoğan, G.; Cordeau, J.-F.; Laporte, G. (2007) *The Pickup and Delivery Traveling Salesman Problem with First-In-First-Out Loading*. Montréal: Elsevier.
- Gjenstrø, A.; Vaksvik, S. (2008) *Single Vehicle Pickup and Delivery Problem with Multiple Commodities and Visit Windows.* Master thesis, Molde University College.
- Glover, Fred; Laguna, Manuel. (1997) *Tabu Search.* Springer, 46-50.
- Gribkovskaia, I.; Halskau, Ø.; Laporte, G.; Vlček, M. (2007) *General Solution to the single vehicle routing problem with pickups and deliveries*. European Journal of Operational Research 180: 568-584.
- Gribkovskaia, Irina; Laporte, Gilbert; Shyshou, Aliaksandr. (2008) *The single vehicle routing problem with deliveries and selective pickups*. Computers & Operations Research 35: 2908 – 2924.

- Kellerer, H.; Pferschy, U.; Pisinger D. (2004) *Knapsack problems.* Springer.
- Klein, Robert. (2000) *Scheduling of resource-constrained projects.* Springer, 198.
- Lee, Kwang Y.; El-Sharkawi, Mohamed A. (2008) *Modern heuristic optimization techniques: theory and applications to power systems.* Wiley-IEEE, 101-122.
- Malapert, A.; Guéret, C.; Jussien, N.; Langévin, A.; Rousseau, L.-M. (2008) *Two-dimensional pickup and delivery routing problem with load constraints.* Montréal: CIRRELT.
- Mladonović, N.; Hansen, P. (1997) *Variable Neighborhood Search.* Computers & Operation Research Vol.24, No.11: 1097-1100.
- Oklobdzija, Vojin G. (2002) *The computer engineering handbook.* CRC Press, 69.
- Ropke, S.; Pisinger, D. (2006) *An Adaptive Large Neighborhood Search Heuristic for the Pickup and Delivery Problem with Time Windows.* Transportation Science Vol.40, No.4: 455-472.
- Toth, P., Vigo, D. (1987) *The Vehicle Routing Problem.* SIAM.
- Vlček, M. (2005) *Heuristic Algorithms for the Single Vehicle Pickup and Delivery Routing Problem and Non-Simultaneous Services.* Master thesis, Molde University College.
- Zouari, Ahmed; Akselvoll, Arne Borch (2007). *Multi-Commodity Pickup and Delivery Supply Vessel Routing for Statoil: Analysis and Modelling.* Master Thesis, Molde University College.