Master Thesis

Development of a Simheuristic Approach for Solving Realistic Inventory Routing Problems

Lena Pfeilsticker
Matriculation Number: 134001
Course: M.Sc. Logistik

Date of Issue:
28-01-2015

Day of Submission:
14-07-2015

Supervisors:

Prof. Dr.-Ing. Markus Rabe
Prof. Dr. Angel A. Juan Perez

Technical University Dortmund
Faculty of Mechanical Engineering
Department IT in Production and Logistics
http://www.itpl.mb.tu-dortmund.de

Universitat Oberta de Catalunya
Internet Interdisciplinary Institute
Department of Computer Science
http://in3.uoc.edu

Abstract

The present thesis introduces a ‘simheuristic’ algorithm, a method combining simulation and metaheuristics, to solve a variant of realistic Inventory Routing Problems (IRPs). The combination of problems of transportational and stock-related issues is complex. Authors in this field focus on certain methodologies with fixed assumptions. This can lead to a restricted applicability for real-life problems. Aiming at improving the applicability of IRPs, an algorithm is introduced based on information provided by real-life IRP companies and extracted from current literature. The goal is to minimize the resulting costs of an IRP system, i.e., the total of inventory and routing costs.

For this purpose, the IRP systems of real-life companies are investigated for such constraints with the goal to develop an approach and subsequently an algorithm based on the result of this investigation and then to implement this algorithm in Java. The model based on the algorithm considers a single-period IRP consisting of multiple depots, customers with stochastic demands and a single warehouse, potential stockouts and a central control unit. Additional constraints are a heterogeneous fleet and multiple products to be distributed. The work finally presents a set of numerical experiments comparing the proposed method with different refill strategies.

To attain this goal the present work utilizes a ‘simheuristic’ approach. As initial situation a stochastic model based on a real-life case is considered, which is then simplified into a deterministic model. A range of deterministic solutions is generated by using the mean values of the stochastic input parameters. If the generated solutions are promising, they are evaluated in a simulation process and ranked in order of the best outcome.

The results of the present work are an identification of characteristics of real-life IRPs, resulting in a simheuristic approach implemented in Java, which is able to deal with realistic requirements of these sorts of problems. The algorithm is tested using current related literature and realistic case studies. A final conclusion of the results is conducted.

Keywords:
Inventory Routing Problem, Metaheuristic, Simulation, Combinatorial Optimization, Real-life Application, Logistics, Stochastic Problems.

Table of Contents

1 Introduction

The field of logistics, transport and inventory management is object of continuous changes due to latest research, changing economies and a changing environment for companies to conduct their business in. In order to stay or become even more cost-efficient and environmentally friendly, companies have to cooperate along the supply chain for improving the efficiency of transport and inventory systems. Independent decision-making processes are a common approach: The partners of a supply chain seek a solution for their own system with the focus on cost efficiency, not taking into account the other supply chain partners (Fraza 1998, Cooke 1998). If the supply chain were to be considered as a whole, an integrated decision-making could lead to significant decreases in total costs and operation durations (Goel and Gruhn 2005 and 2008). An approach to this problem implemented in many systems is Vendor Managed Inventory (VMI), often used e.g., in the retail industry (Waller et al. 1999). A centralized control unit of a supplier with access to the inventory levels and demand of a company is fully in charge of the replenishment. Thus, the supplier can fully adjust transportation and inventory levels and provide more efficient services, compared to the two processes being considered separately. The general problem of integrating inventory and transportation (vehicle routing) decisions is called an inventory routing problem (IRP) (Campbell et al. 1998). The majority of authors dealing with a problem in this research area either address the inventory problem or the transportation problem. Less published works deal with the integrated decision-making of inventory and transportation (Golden et al. 2008). This is due to the fact that the necessary procedures and models to obtain a solution to such a problem are very complex and can take a long time to process. Considering the reality of supply chain systems and real-life IRP applications, it is necessary to conduct research by considering the whole logistical system to find optimized solutions for minimal cost and highest efficiency. A significant progress in this research field has already been reported by Juan et al. (2014a). The authors propose an approach combining simulation and metaheuristic to a so-called ‘simheuristic’, leading to solutions which are improving known solutions to IRPs. In the present work an approach is developed that aims at a better applicability to realistic rich IRPs.

The general objective of the present work is to develop a simheuristic approach for solving a rich IRP by including uncertainties and realistic constraints extracted from information in literature and provided by companies, while minimizing the overall costs. The main challenge and another goal is to find a way to include these realistic problems in the approach, considering that the problems can vary widely and that information provided by companies and extracted from current literature might be very little, incomplete or not relevant for the present work.

By using the simheuristic approach it is possible to quantify the distribution of risks related to stochastic customer demands when pre-defined refill strategies are applied to the suppliers’ inventories. The objective is to produce an evaluation of risks connected to different stochastic scenarios in customer demand.

Ultimately, after a verification and validation process, the implemented algorithm should be able to use real-life data to solve specific IRPs.

The structure of the present work is as follows: After an introductory part, the characteristics of real-life IRPs are extracted from current literature and from information provided by several companies. Following this, the information is translated into requirements for the formulation of the approach used for the treated IRP. A specific problem description is given and a solving process proposed, explaining how the IRP benefits from the simheuristic approach. Afterwards a pseudocode is created, containing the description of all necessary steps, inputs and outputs. Subsequently, an algorithm based on the developed approach is implemented in Java in several steps. The intermediate results and the final code are verified and validated in various tests. Concluding the chapter, a risk analysis is conducted to demonstrate the costs resulting from different refill policies and different fluctuations in customer demands. Then an exact description of the final code is given and its functionalities are explained. Finally, a conclusion is completed to discuss the results.

2 Theoretical Groundwork for realistic IRPs

In this chapter a comprehensive literature research is given with the aim of introducing the subject of vehicle routing problems (VRPs) and especially IRPs. Ensuing this, the methods applied in the present work are introduced and explained.

2.1 Overview Vehicle Routing Problems

IRPs have been developed as a separate form of VRPs for several decades until now. The requirement for extended research on VRPs was launched by companies' needs for more applicable models of their realistic systems which led to the research area of IRPs (Coelho et al. 2014). In general the solving of a VRP includes optimizing a routing system, without violating given constraints, while satisfying all of the customers' demands. The objective function is to minimize the overall costs (Toth and Vigo 2002). IRPs were usually variations of modeled VRPs, where additional constraints such as inventory costs were taken into consideration (Coelho et al. 2014). The research field of VRPs was first introduced as a generalization of the travelling salesman problem (TSP) by Dantzig and Ramser (1959). Since then, a lot of variants have been developed and new branches of VRP research lines were combined to form new research fields. (Caceres et al. 2014). The main variants of the VRP can be found e.g., in Golden et al. (2008) or Caceres et al. (2014). The modifications depend on the considered parameters and constraints. They can vary widely and can be combined to form new problems, either with interdependencies or without (Caceres et al. 2014). Among the most common variants are the Capacitated VRP (CVRP), where each customer has a demand for a product and vehicles have finite capacity; the VRP with Time Windows (VRPTW), where each customer must be visited during a specific time frame; the VRP with Pick-up and Delivery (PDP), where goods have to be picked-up and delivered in specific amounts at the customers; and the Heterogeneous fleet VRP (HVRP), where vehicles have different capacities. (Pillac et al. 2013)

Even though VRPs have been studied for decades, the research around this problem type is extended constantly. One of the latest approaches is increasingly, to include real-life problems or attempt to solve more realistic types of VRPs by combining multiple constraints. This led to considering specific combinations of real-life constraints and to Rich VRPs (RVRPs). This type of problems deals with realistic optimization functions, uncertainties and dynamism of inputs while including a variety of real-life constraints. (Caceres et al. 2014) Following this, IRPs can be seen as a type of RVRPs.

In most current research the definition of RVRPs is not clear since there is no formal definition or criterion which identifies a problem clearly as such (Lahyani et al. 2015). A first attempt to define the RVRP was made by Toth and Vigo (2002). The authors define the potential of extending the "vehicle flow formulations, particularly the more flexible three-index ones." Lahyani et al. (2015) propose the following definition: "A RVRP extends the academic variants of the VRP in the different decision levels, by considering additional strategic and

tactical aspects in the distribution system (4 or more) and including several daily restrictions related to the Problem Physical Characteristics (6 or more) [pure routing or operational]. Therefore, a RVRP is either a VRP that incorporates many strategic and tactical aspects and/or a VRP that reflects the complexities of the real-life context by various challenges revealed daily."

Authors like in Lahyani et al. (2015) attempt to provide a generic taxonomy for the RVRP literature and a clear definition. After reviewing numerous current research works, they state a RVRP as an extension "of the VRP in the different decision levels by considering at least four strategic and tactical aspects in the distribution system and including at least six different daily restrictions related to the physical characteristics. When a VRP is mainly defined through strategic and tactical aspects, at least five of them are present in a RVRP. When a VRP is mainly defined through physical characteristics, at least nine of them are present in a RVRP."

Physical characteristics include restrictions like transport weight, number and type of vehicles, travelled distances, number and type of products etc. Strategic and tactical aspects include e.g., the transportation strategy, number of depots used or visit frequencies of customers.

Authors like Caceres et al. (2014) refer to RVRPs reflecting "as a model, most of the relevant attributes of real-life vehicle-routing distribution systems. These attributes might include dynamism, stochastic, heterogeneity, multi-periodicity, integration with other related activities (e.g., vehicle packing, inventory management, etc.), diversity of users and policies, legal and contractual issues, environmental issues, and more. Thus, as a model, an RVRP is an accurate representation of a real-life distribution system and, therefore, the solutions obtained for the RVRP should be directly applicable to a real-life scenario."

To conclude: the presented authors include numerous characteristics and constraints in the definition of RVRPs in order to ascertain a certain level of complexity (and uncertainties). This pattern can be applied to IRPs as well, being a type of RVRP. In the next chapter the research field of IRPs is introduced

2.2 IRPs in the realistic environment of uncertainties

The IRP in current literature is commonly described as a problem "integrating inventory management, vehicle routing and delivery scheduling decisions" (Coelho et al. 2014). Companies involved in this problem type have to make simultaneous decisions about when to serve a customer, how much to deliver to this customer when served and how to combine customers into vehicle routes. When it comes to solving IRPs, a lot of different approaches have been developed.

Several studies have combined simulation and optimization approaches to find solutions to complex optimization problems (Augerat et al. 1998, Andradóttir 2006, Anily and Federgruen 1993). The supply chain process has been a popular target for this type of technique. Various works focus on channel coordination (Eskandari et al. 2010), scheduling problems (Angelidis et al. 2012) or inventory problems (Alizadeh et al. 2011) using simulation-optimization modeling. In the past years, several approaches have been proposed for different variants of the Inventory Routing Problem.

The main factors to take into account when classifying the different works are

(a) whether they consider deterministic or stochastic demands;

(b) whether they consider single or multiple periods (including an infinite horizon);

(c) whether they allow inventory shortages or not;

(d) whether they consider single or multiple products;

(e) whether they use the same refill policy for all nodes or specific replenishment policies for each node; and

(f) whether they use exact or approximate methods to solve the problem.

The following literature review is divided according to the first – and probably most relevant – criteria, i.e., whether the demands are considered to have a deterministic or a stochastic nature.

The IRP with deterministic demands

Regarding the IRP with deterministic demands, various modeling and solution approaches have been proposed, the majority of which are summarized in Bertazzi et al. (2008). Some use integer or dynamic programming approaches, e.g., Chien et al.(1989), Campbell et al. (2002), and Campbell et al. (1998), while others utilize a heuristic or metaheuristic approach, e.g., Anily and Federgruen (1990) and Bramel and Simchi-Levi (1995). Other sequential approaches include Campbell and Savelsbergh (2004a) who present a two-phase approach including support of GRASP-like randomization 'as a powerful tool to improve the performance of insertion heuristics'. Campbell and Savelsbergh (2004b) and Campbell and Savelsbergh (2004c) consider extensions. Mjirda et al. (2014) propose a two-phase Variable Neighborhood Search metaheuristic to solve a multi-product IRP with deterministic demands over a finite planning horizon.

The IRP with stochastic demands

One of the first works on the IRP with stochastic demands is Federgruen and Zipkin (1984). The authors adress the single-period combined problem of ''allocating a scarce resource available at some central depot among several locations, each experiencing a random demand pattern, and planning deliveries using a fleet of vehicles''. Trudeau and Dror (1992) address a multi-period version of the IRP with stochastic demands in which stockouts are also allowed. Godfrey and Powell (2002) describe an adaptive dynamic programming algorithm with stochastic inputs and a single-period setting. Barnes-Schuster and Bassok (1997) address an infinite-horizon scenario in which demands are stochastic. Jaillet et al. (2002) solve an IRP in a rolling horizon framework with stochastic demands and allow for inventory shortages. Reiman et al. (1999) consider an IRP with stochastic demands and one vehicle covering a region composed of several customers comparing three strategies. Gaur and Fisher (2004) describe a similar application with stochastic demands and a heterogeneous fleet. In Yu et al. (2006), the authors analyze the multi-period stochastic IRP with split delivery, aiming at transforming the stochastic model into a deterministic one. Juan et al. (2014a) use an individual policy for each retail center in their single-period IRP with stochastic inputs. They solve a single-period IRP with a homogeneous fleet, one depot and possible stockouts by applying an approach combining simulation and metaheuristic.

2.3 Methodology

When dealing with a problem type like VRPs, particularly IRPs as a specific case, the methodology has to fulfill a set of demands to be able to be successfully applied.

Caceres et al. (2014) propose a classification based on the level of complexity and uncertainties. It is displayed in Figure 1.

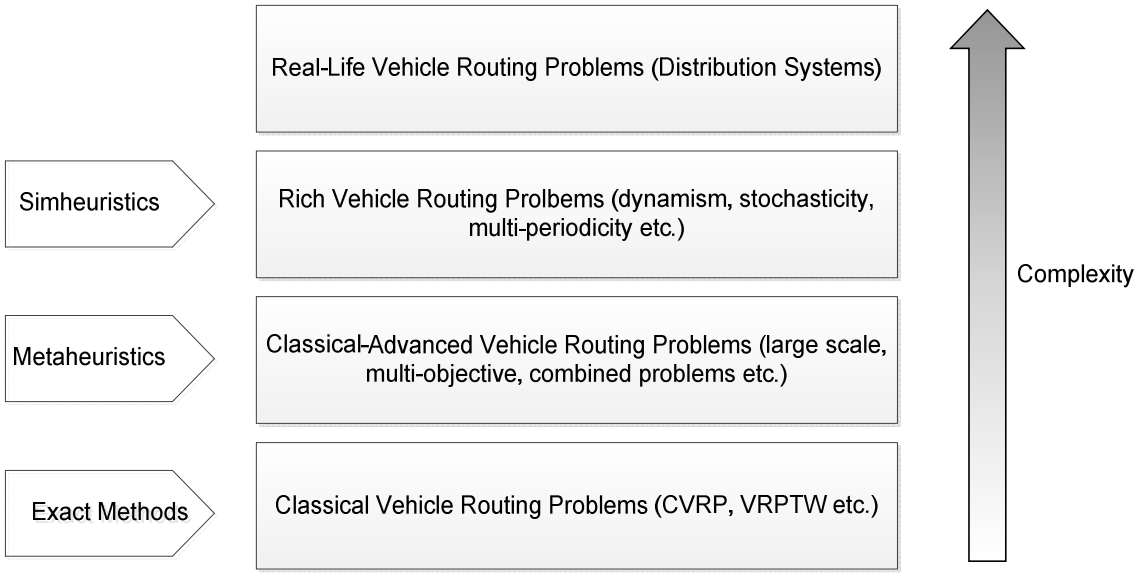

Figure 1 *Classification of Applicable Methodologies (based on Caceres et al. 2014)*

The classification shows a need for increasing flexibility of the applied methodologies when the level of complexity increases. While classical VRPs can be solved with exact methods in reasonable computing times, the number of approximate solving parts of methodologies has to increase when more constraints are added to a type of VRP. Since with an increasing number of combinatorial elements, the problem size increases exponentially (Juan et al. 2010a), exact methods have too extensive computing times. Also, since this kind of problem is typically NP-hard, the use of exact algorithms is limited and metaheuristic concepts are advantageous (Schmid et al. 2013).

Several studies have combined simulation and optimization approaches to find solutions to complex optimization problems (van Dijk and van der Sluis 2008, Laroque et al. 2012, Gonzalez et al. 2012). This approach is adapted and extended in the present work.

In the following chapter the methods applied to the here treated IRP are explained.

2.3.1 Simheuristics

The methodology of the present work consists mainly of two combined approaches: simulation and metaheuristics. Linked together they form a new methodology called ‘simheuristics (Juan et al. 2015).

In the present work the solving process for the given problem is embedded in a simheuristic framework. It is able to build an interface between real-life stochastic IRPs and deterministic

methods commonly used for complex problems in this research area (van Dijk and van der Sluis 2008). Simheuristics are optimization-driven evaluation functions used for NP-hard, combinatorial optimization problems (COPs). They use simulation to obtain good feasible solutions in reasonable computing times. (Juan et al. 2015)

Metaheuristics are a class of solution methods using approximate optimization techniques for solving complex problems in various research fields. Metaheuristic models can be optimized and implemented into algorithms, which makes them easily applicable to many different kinds of optimization problems. They are part of heuristic algorithms but are not problem-specific and thus can be varied and adapted to different specific scenarios. Metaheuristics provide acceptable solutions in a reasonable computing time when solving complex problems. (Talbi 2009)

Simulation approaches enable to model and study complex systems with various impacts by varying input parameters and processes. Simulation models have underlying assumptions and solve problems with the help of stochastic elements in 'appropriate' computing times. They are models of real-life systems and thus can be used to test existing systems or systems in development without risks. (Juan et al. 2015)

Realistic IRPs are a type of problem with a high level of complexity and, when combined with stochastic values, with a high number of uncertainties. When solving complex logistical problems, exact solutions take long computing times (Juan et al. 2014a) due to many elements with several characteristics which have to be combined to find a feasible solution. Adding an element can lead to a significant rise in possible combinations and thus the problem size to solve is increasing exponentially (Juan et al. 2010a). Therefore, for this problem type, a heuristic approach is sufficient to achieve an approximate solution in a much smaller time frame. Heuristics are applied to find near-optimal solutions. When dealing with realistic problems, solutions need to be found in a reasonable time so they can be applied.

When dealing with heuristic approaches, there is no optimal solution or solving process. In order to find reasonably good and feasible solutions, different combinations of elements have to be tried out and existing solutions need to be improved. One way of doing this efficiently is simulation. It enables to test different scenarios and calculate approximate costs and operation times.

When applying the simheuristic, a stochastic model of a real-life IRP is constructed with several constraints. It is converted into a deterministic model by using the expected mean values of the single input parameters. The deterministic problem is solved using a heuristic approach and the solution is tested for feasibility. In a fast simulation process with a low number of replications the promising solutions are evaluated by using stochastic parameters, leading to a stochastic solution to the COP. The generated solutions are ranked according to their best outcome and the best results are again tested for their feasibility and quality in an intensive simulation process with a large number of replications to determine the best solution (Juan et al. 2015). The final results are feasible solutions with the best outcome of the system in a given maximal computation time. This process is displayed in Figure 2.

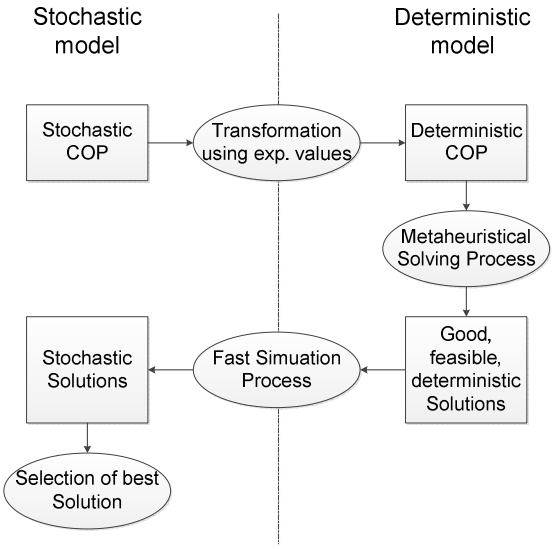

Figure 2 *Simheuristic Approach (based on Juan et al. 2015)*

During the process of fast simulation, the best computed solutions are saved in a cache. When the iterative process of finding solutions is repeated, the new computed solutions are compared to the already computed best solutions in the cache. If the solutions in the cache are worse, the newly computed solution replaces the worst solution in the cache. This way, promising solutions are selected over inferior ones. Using this approach does not guarantee finding the overall best solution for the problem it is applied to. But, it gives a good chance finding reasonably good solutions in a given maximum time.

The final simulations hold the possibility of evaluating the quality of each solution and do a risk analysis showing not only the expected result of the chosen solution but also the distribution of the alternative solutions involved. Thus the decision makers can evaluate if a solution with an expected better outcome but higher risk of failure can be less attractive than a solution with a lower risk but a still satisfying outcome.

As illustrated, the advantage of combining metaheuristics and simulation approaches lies in providing solutions modeling complex real-life problems with a high flexibility and quality within reasonable computing times.

2.3.2 Biased Randomization

During several steps of the solving process a biased randomization is applied. This describes the use of nonsymmetrical probability distributions to generate randomness in heuristics (Juan et al. 2013). It can also be seen as random sampling from non-uniform distributions in a Monte Carlo simulation.

Heuristics are applied to find near-optimal solutions in an iterative process. During the application of the heuristic, elements to construct the next movement are chosen from a previously sorted list. The elements on the priority list are sorted according to a criterion of the highest advantage for the solution. E.g., for an IRP a list of customers with the closest location to a depot is computed and the depot gets assigned the customers with the top positions on the priority list, i.e. the customers with the minimal distance to the depot. However, heuristics are

deterministic approaches: using one input will always have the same results. To optimize the simulation and obtain several solutions from which the best solution can be selected, the elements of the priority list are combined with biased randomization.

This means, a probability to be selected is matched with every element of the priority list, generating a stochastic approach. In the following simulation, the order, in which the elements of the priority list are selected, is different at every step of the iterative solving process and multiple solutions are obtained. A uniform distribution of the priority list's elements would destroy the logic of creating a priority list in the first place. In order to avoid this, a biased randomization is employed, assigning higher probabilities to be picked to elements with higher positions on the priority list. This approach has been proven to outperform deterministic approaches (Juan et al. 2013). The difference between uniform and biased/non-uniform randomized priority lists is displayed in Figure 3.

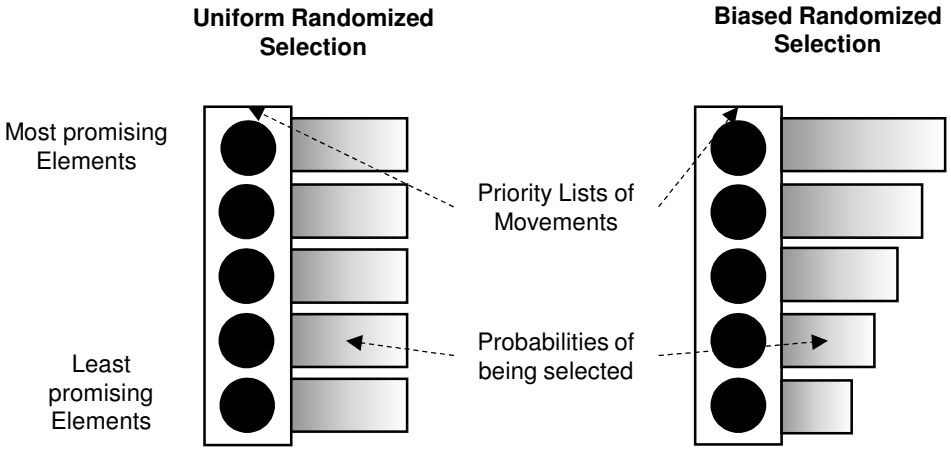

Figure 3 *Biased Randomization of Priority Lists (based on Juan et al. 2015)*

Biased randomization is applicable whenever a new order of given elements needs to be constructed based on a sortable criterion. The order of the elements can be randomized and thus new solutions can be computed, based on a logical order. When solving an IRP, a lot of elements need to be arranged and biased randomization favors the beneficial choices during the process. Due to this functionality, biased randomization is applied during the solving process and serves to improve computed solutions.

2.3.3 Multi-Start Approaches

Multi-start (MS) approaches conduct solving processes by applying local search to start from different initial solutions, initiated by a repeated constructive process. The classical Random Restart procedure and the more recent GRASP procedure (Festa and Resende 2002) are well-known examples for MS approaches (Fleurent and Glover 1999).

Heuristic search procedures provide approximate solutions to highly complex problems. They are based on pre-defined assumptions limiting the search space of solutions. One way to find the optimal solution in this search space is to calculate global extrema. Usually this type of heuristic approaches re quires some type of diversification to overcome local optimality. In

literature and research, the most successful approaches have two phases that are alternated for a certain number of global iterations. During the first phase an initial solution is computed. In the second phase this solution is improved. Each global iteration produces a solution that is typically a local optimum, and the best overall solution is the output of the algorithm. The interaction between the two phases creates a balance between search diversification (structural variation) and search intensification (improvement), to yield an effective means for generating high-quality solutions. Following a survey of MS methods for COPs, the authors distinguish between memory-based and memoryless procedures. (Martí et al. 2013)

The authors in Glover (1977) introduce a framework in which MS search includes local search to improve the starting solutions. Within this framework, procedures are given for generating starting values for variables and for generating values perturbed from other starting points. By varying the rules for the perturbation, these strategies include customary local search approaches for producing re-starts. A series of extensions of this framework are given in Glover (1986, 1989, 2000), addressing controlled randomization, learning strategies, induced decomposition, and adaptive memory processes (as introduced in tabu search). Emphasis is placed on the interaction between intensification and diversification as an instrument for creating a more effective search process.

2.3.3.1 Iterative Local Search

Iterated Local Search (ILS)is a special case of a MS approach. Among the metaheuristics for NP-hard optimization problems, the ILS in current research is seen as one of the effective and simple approaches to find solutions. It has been successfully applied in many combinatorial optimization problems. Martin et al. (1992) first introduced the 'kick' method, which was later referred to as iterated local search. Lourenço et al. (2001) presented a beginner's introduction to iterated local search. ILS can help the local optimizer escape being trapped in a local minimum while keeping numerous useful properties of the local minimum. This can make ILS more efficient and effective than randomly restarting a local search. (Tang und Wang 2006)

During the solving process, an ILS is applied to individual elements. It is used in computer science as a modification of a local search process when solving optimization problems.

Local search methods are used when a solution in a search space can be found by determining the extrema of the related mathematical function. The responding algorithm moves from solution to solution in the search space by applying local changes. This process continues until a determined termination criterion is met, e.g. elapsed time or surpassing a previously set value limit. These algorithms are commonly applied to hard computational problems, like the travelling salesman problem. (Hoos and Stützle 2005)

When calculating extrema of a mathematical function, a local search can get stuck in a local extremum, where no improving neighbors are available. Lourenço et al. (2002) introduce an approach modifying the local search by iterating the routine. Each time the process is repeated, the starting point is varied. This is not to be confused with the usage of a memory, meaning the knowledge obtained during previous iterations is not used for the next iterations. It is a repeated local search. The authors define it as follows: "one *iteratively* builds a sequence of solutions generated by the embedded heuristic, leading to far

better solutions than if one were to use repeated random trials of that heuristic" Lourenço et al. (2002). They also state that the characteristics of an ILS include "(i) there must be a single chain that is being followed (this then excludes population-based algorithms); (ii) the search for better solutions occurs in a reduced space defined by the output of a black-box heuristic."

When applying the method, the authors state that an ILS explores heuristically a given search space using "a walk that steps from one solution to a 'nearby one', without the constraint of using only nearest neighbors". Given a current solution s*, the solution is partially perturbated which leads to an intermediate state s' of the solution. Then, a local search is applied to s' and a new solution s*' is determined. All the states and solutions are part of the search space. If s*' meets an acceptance criterion, it becomes the next element of the walk in the search space, otherwise, one returns to s* for searching another new solution. The resulting walk is a case of a stochastic search in the given search space. This process is shown in Figure 4.

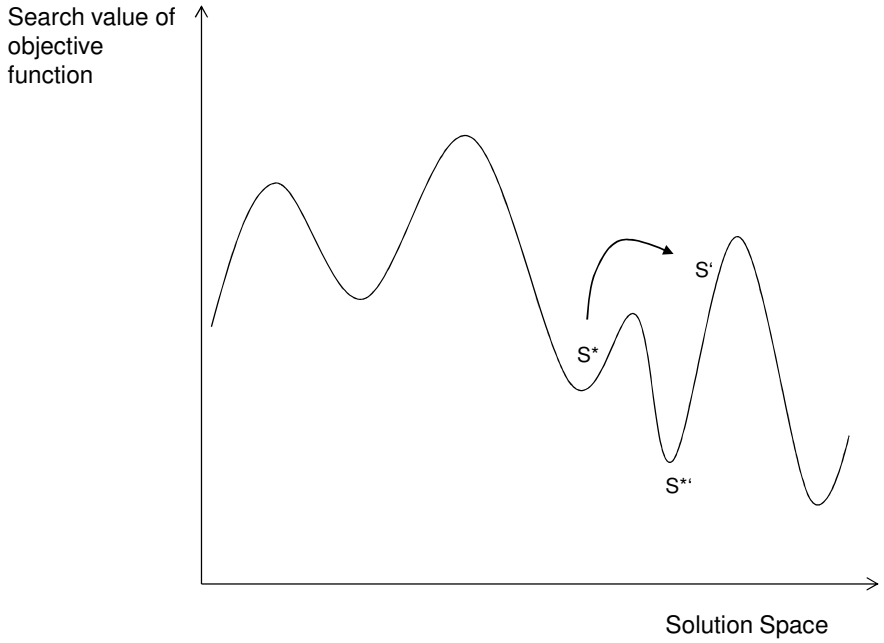

Figure 4 *Iterated Local Search Process (based on Lourenco et al. 2002)*

Starting with a local minimum s*, a perturbation is applied leading to a solution s'. After a local search, a new local minimum s*' is found that may be better than s*.

When applying an ILS, the implicit assumption is that of a clustered distribution of local minima: when minimizing a function, determining reasonably good local minima is easier when starting from a local minimum with a low value than when starting from a random point. It is necessary to avoid getting stuck in a given attraction basin. The distance between two solutions needs to be big enough to transform the current solution to a reasonable starting point.

The perturbation strength has to be sufficient to lead the trajectory to a different attraction basin, which leads to a different local optimum. In the optimal case a global optimum is found.

In the present work, an ILS is applied during the process for fixedly assigning customers to depots from which they are supplied (see chapter 3.3). It serves to explore feasible solutions for a given problem and finding the best possible option that can be found in a given time frame.

2.3.4 Round-robin Process

A round-robin approach is used to choose elements in a rational order. Usually when it is the element's turn, a similar share of resources is allocated to the element. Generally, when employing a round-robin approach, equal shares of a limited resource are distributed to each process in a circular order. The processes are not prioritized and no process is cut off of the available resource. The name of the algorithm comes from the round-robin principle known from other fields, where each person takes an equal share of something in turn.

In computer science it is employed to allocate limited resources to competing processes, usually time. The processes are suspended after their allocated time is up. (Silberschatz et al. 2010)

The process is displayed in Figure 5.

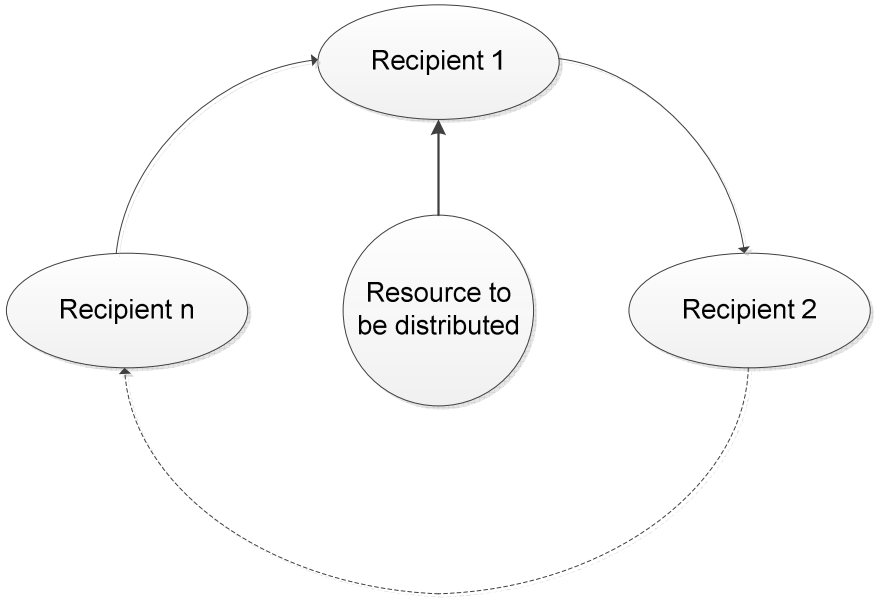

Figure 5 *Round-robin process*

In the present work the round-robin scheduling is used during the assignment of customers to depots. The authors of Juan et al. (2014b) use this approach with three different variations: "(a) a round-robin tournament criterion following consecutive turns among depots is used to guarantee that a different depot selects a new (customer) at a time – as far as it still has enough capacity to serve the associated demand; (b) the same round-robin criterion, but this time a depot, is randomly selected at each round for the node-selection turn; and (c) at each round, the depot with the most (remaining) serving capacity selects the next node from its priority list."

The authors state that the capacitated round-robin process (c) leads to reasonably good "balanced" allocation maps of customers to depots as compared to the other proposed

possibilities. Due to these results, in the present work a capacitated round-robin approach is employed.

2.3.5 Clarke and Wright Savings Algorithm

The Savings Algorithm (CWS) was first introduced in 1964 by Clarke and Wright and is today one of the most practiced approaches for solving routing problems in logistical systems, e.g., the TSP (Clarke and Wright 1964, Suhl and Taieb 2009). It is a heuristic approach for approximate solutions in a reasonable time.

The given problem is to find the shortest distance between a starting and an endpoint (called depot in the following) with nodes (customers) in between or to generate the highest savings possible. Furthermore, it is necessary to determine the allocation of the customers among routes, the sequence in which the customers shall be visited on a route, and which vehicle shall cover a route. The objective is to find a solution which minimizes the total transportation costs. More constraints are that that every customer is visited exactly once, where the demanded quantities are supplied and the total demand on every route must be within the vehicle's capacity.

The algorithm assumes an initial solution where every node is directly connected to the depot with a route going there and directly back. When implementing the problem, a necessary presumption is that an unlimited number of vehicles, with sufficient capacities, is provided to form the routes building the initial solution. Another presumption is a symmetrical distance matrix.

When solving the algorithm, the determination of the customers being visited on a tour and the order, in which they are visited, is simultaneous.

In the following, the single steps for the algorithm are described:

1. Each node is connected with the depot with a route directly to and from each customer.
2. Now, an edge of two customers respectively from/to the depot is detached and the customers and the depot are reconnected in one route. This is done for all possible combinations of customers.
3. Now, the savings for the combinations are calculated for all combinations:

$$\begin{aligned} Savings \\ &= distance\ from\ customer\ i\ to\ depot \\ &+ distance\ from\ customer\ j\ to\ depot \\ &- distance\ between\ customers\ i\ and\ j \end{aligned}$$

4. All saving values are sorted in a descending order.
5. The two nodes with the highest savings value are connected in a new route, provided both nodes still have one edge with the depot.
6. Step 5 is repeated until only two edges from/to the depot are left.
7. When only two edges from/to the depot exist, the solution is reached. This solution is feasible, provided that the vehicle's capacity covering this route is not exceeded by the customers' demands on the route.

(Clarke and Wright 1964)

The procedure of generating savings is displayed in Figure 6.

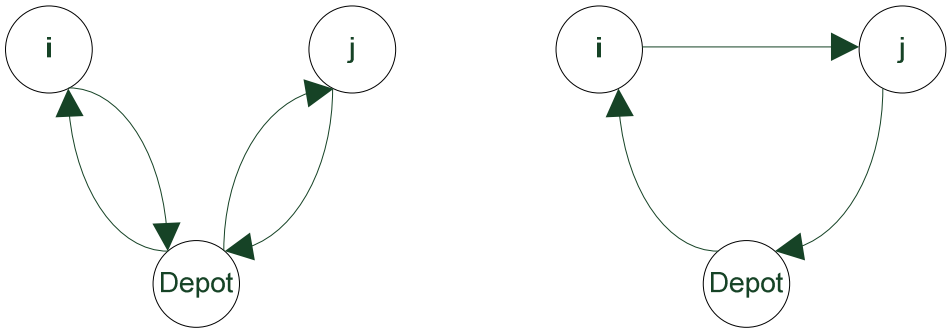

Figure 6 *Clarke and Wright Savings Algorithm (based on Clarke and Wright 1964)*

The first picture shows the initial solution with two customers i and j. In the second picture on the right, savings have been generated by combining the two customers in one route from and to the depot.

3 Development

3.1 Analysis of Requirements

To determine which constraints are applied to the IRP treated in this work, an analysis of current literature and on information provided by several companies is conducted. The characteristics qualified for being chosen as requirements for the treated IRP should not be completely new to the field but have been considered before in other research, to be sure that an impact on the IRP exists. It would be optimal if the characteristics were used in implementations before, to obtain a first idea on how to implement the approach treated here and to read about the results of previous implementations.

Several authors (see chapter 2.2) choose a low number of characteristics for their IRPs due to the high complexity of the ensuing solving process. These assumptions do not necessarily reflect the reality of IRPs in companies or supply chains. Since one goal of the present work is to solve a realistic IRP, a higher number of characteristics are chosen as requirements and constraints as long as they do not contradict each other. In order to present a solution to a new IRP, the requirements are chosen in a combination that is not similar to characteristics in other authors' works.

When choosing requirements, it is beneficial when the information provided by companies and the results from the literature research match. This has the advantages of using literature as described above and that for some of the characteristics a set of real data is possibly available which can be used as input to test the developed program.

Finally, characteristics that are already used in Juan et al. (2014a) are included or extended and applied to the treated IRP here. The constraints and requirements in the IRP of Juan et al. (2014a) are a single-period, single-product, single-depot, homogeneous fleet, centralized control unit, possible stockouts, stochastic demands and no time windows for the deliveries. A comprehensive overview of characteristics of IRPs in current literature is given in chapter 2.2. Based on this literature research, the main factors, as listed in chapter 2.2, are taken into account when classifying the different works. The descriptions of realistic IRPs provided by companies have been analyzed and characteristics applying to IRPs are listed as follows:

- Consideration of multiple periods or single periods (due to a changing range of customers to be supplied every day)
- Multiple depots
- Warehouses additionally to depots
- Multiple products
- Time windows for deliveries
- Heterogeneous fleet
- Possible stockouts
- Centralized or decentralized control units for stock replenishments
- Stochastic demands

After reviewing all proposed characteristics and a special check for the accordance of them in all considered sources, a combination of several requirements is intended to reflect the reality of IRPs sufficiently while at the same time being implementable in a code.

Based on the literature review in chapters 2.1 and 2.2 and the analyzed information provided by real-life companies, the requirements for the treated IRP are set as follows: It is considered with multiple products, multiple depots, one warehouse, a heterogeneous fleet, a single period, stochastic demands, a centralized control unit, possible stockouts and without time windows for the deliveries.

In the following chapter a complete problem description is given.

3.2 Problem Description

In the present work the considered IRP is a single-period, multi-product, multi-depot IRP with a heterogeneous fleet and stochastic demands. The decision about the refills of the depots is made by a central control unit. Stockouts are possible. The IRP is stated as follows:

A company supplies four different types of products to their 1,000 customers. Each customer has deposits for their one or multiple demanded products. The distribution of the four products is conducted from three depots and one warehouse which has a virtually unlimited capacity of every product but belongs to another company. There are no safety stocks at the depots.

Each depot holds an inventory level for every product with defined individual and overall capacities. These capacities are determined by a set refill policy that is calculated based on the expected customer demand for every depot.

The vehicles used to distribute the products are based at the depots. The fleet is fixed and heterogeneous, each vehicle with a fixed total capacity. They have a number of different compartments to hold different products. However, one compartment cannot hold different products and must be emptied before being filled with another one. But, every compartment can be filled with any type of product.

The vehicles are divided in small, medium and large ones. Only the medium and large sized vehicles are able to refill their compartments at the warehouse. The vehicles driving to the warehouse must be empty. Each vehicle is assigned to one depot. It is always the point of departure and return.

The capacities of the vehicles are as follows:

- Small vehicle: 4,000L overall capacity with two compartments (50%/50%)
- Medium vehicle: 9,000L overall capacity with three compartments (20%/35%/45%)
- Large vehicle: 30,000L overall capacity with three compartments (20%/35%/45%)

Each depot has a fixed number of vehicles assigned to it, depending on the amount of products it has to serve per period.

An exemplary system is displayed in Figure 7

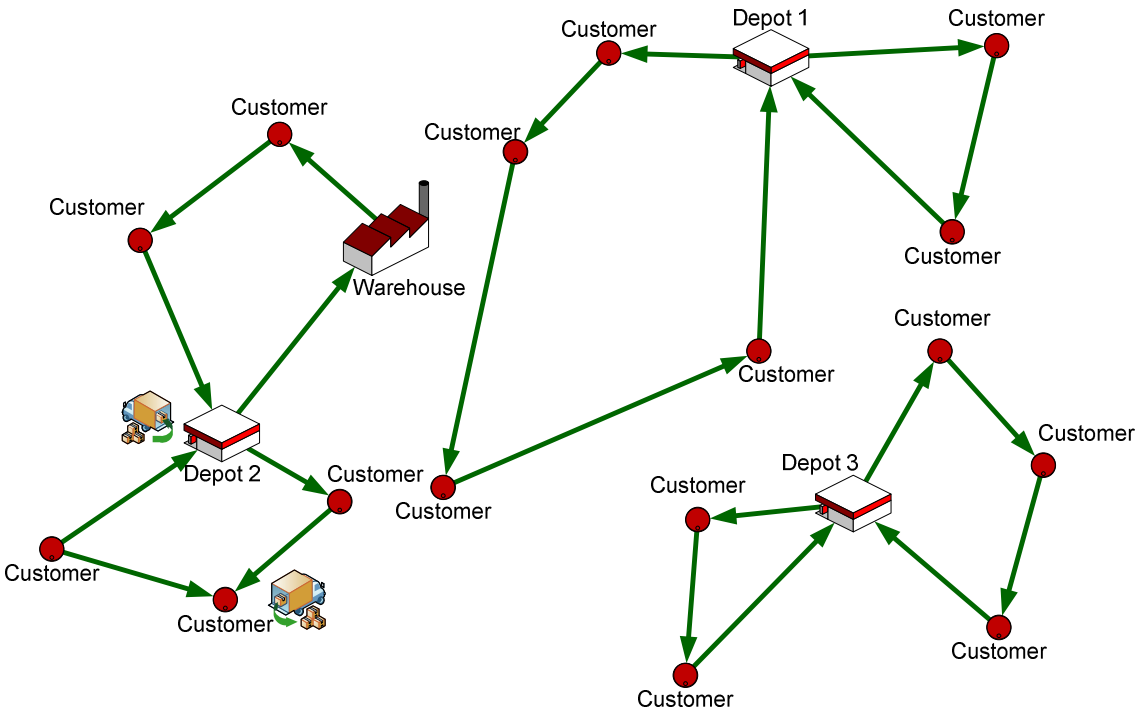

***Figure 7** Inventory Routing System*

The ordered amount of products of each customer is known at the beginning of every period. However, upon arrival of the vehicle, the customers have the possibility to change their ordered demand and thus create stochastic demands. This can lead to route failures due to insufficient amounts of loaded products on the vehicle. If this is the case, a second vehicle has to return to the customer with the missing number of products.

The cost function for this system consists of three components:

- Set-up costs: each shipment, with loading and unloading products, has associated fixed costs.
- Transport costs: based on the kilometers traveled and the volume transported.
- Storage costs: the costs of maintaining a sufficient level of inventory while not storing more than needed to reduce costs. This also includes maintenance and facility costs.

The goal is to minimize the cost function while satisfying 100% of the customers' demands in every period.

In the next section a solving process for this IRP is proposed.

3.3 Proposed Solving Process

To find a solution to the described problem, the proposed solving process is divided into three steps: assignment, inventory planning and routing. The goal is to find the overall minimum costs. The process described in this chapter is extended with more details in chapter 4, when the implementations are described. This is due to the fact that it is not clear to which details attention needs to be paid to or which details have not been considered yet at this stage. The implementation of the described problem is an iterative process and can lead to significant

changes in the final solving process as opposed to the proposed one. The solving process is a combination of developed methodologies which are then extended to suit the problem in the present work.

The inputs for the computation of the solution are as follows: location of deposits, location of warehouses, location of customers, distribution of random demands, current inventory levels for every product, possible refill policies, inventory-cost function, routing-cost function, number of vehicles by type and their associated number of compartments.

3.3.1 Assignment

This step fixedly assigns customers to depots, creating a single-depot IRP for each depot by considering the depot and its assigned customers as a closed system. First, a priority list of nodes is generated for every depot. The list is based on marginal savings in distance costs and considers the maximum capacity of the depots with regard to the combined customer demand for every product. That means the customers are prioritized when they are closest to a depot in comparison to all other existing depots and are assigned while the overall capacities of products in this depot are not exceeded. This is a deterministic approach which only allows one possible outcome. To obtain multiple solutions for comparison of the best outcome a biased randomization is introduced to assign customers to depots (see chapter 2.3.2). This approach assigns higher probabilities to nodes with a better expected outcome without destroying the basic logic of the heuristic approach.

Then, a first solution is generated using a round-robin process. At each round of the round-robin process, the depot with the most (remaining) overall serving capacity selects a node from its generated priority list. This process is displayed in Figure 8.

During this step the capacity constraint for the depot is a soft constraint and can be violated to a certain extent. The assumption is that the customers can not only be served from the depot but also from the warehouses with unlimited capacity. This implies that the capacity for every product, which can be served to the customers from each depot, is determined by the combined capacities of every vehicle assigned to this particular depot. A vehicle can start its route at one of the warehouses (first stop), arrive empty and then proceed to deliver products to customers. This consideration differs from the work of Juan et al. (2014a).

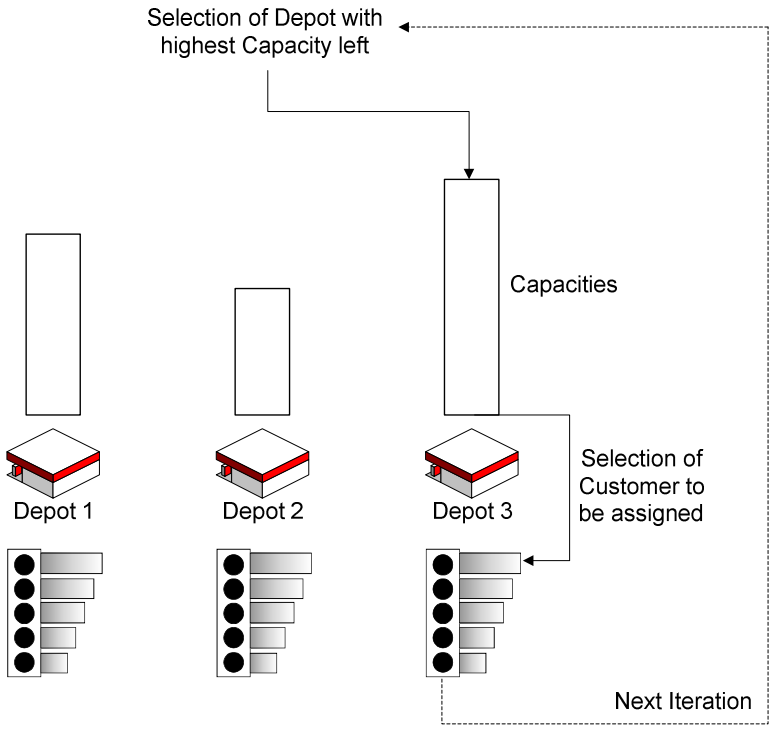

Figure 8 *Capacitated Round-robin process*

After obtaining a first solution, an ILS generates thousands of maps with feasible solutions of assignment (customers to depot) in a short time. This is done by perturbating the base map in a construction-destruction process.

The result is a single-depot IRP for each depot. The routing is done separately for every depot and its assigned customers for each of the generated maps of the assignment step. This separate consideration of every depot has the advantage that the number of possible combinations of consecutive customers is decreased. This procedure is based on a splitting policy. By splitting the multiple-depot IRP into a single-depot IRP the size of the problem decreases exponentially and thus results in shorter computing times for finding feasible solutions. (Juan et al. 2010a, Juan et al. 2010b)

3.3.2 Inventory Planning

For each of the depots, different refill policies are tested to determine the inventory levels with the lowest costs. For this purpose, the accumulated customers' demands of every depot are calculated to estimate the overall demand of every product for every depot. This is performed based on the results from the previous assignment step.

Then the expected inventory costs associated with each combination of depot and refill policy are calculated, including stockouts. With a basic CWS heuristic the routing costs resulting from each refill policy are calculated. For this purpose an initial solution is computed where one vehicle drives directly between one customer and the assigned depot.

The policy with the lowest overall costs is chosen to serve for a computation of an initial base solution for the routing process which is described as follows.

3.3.3 Routing

The last step generates routes from every depot to each of its assigned customers, employing a biased randomized CWS heuristic. To do this, a first initial solution is necessary (see chapter 2.3.5), which is computed based on the determined refill policy in the previous step. To realize the implementation process, an unlimited number of the smallest trucks available, is provided to form routes directly to and from the depots to every customer assigned to it.

Then, the biased randomized CWS heuristic is performed to improve the first solution (Juan et al. 2010a). While performing the CWS heuristic, the vehicle with the biggest surplus of overall capacities is chosen first, accounting for its number of different compartments and their individual capacity as opposed to the customers' demands. When merging the single routes, the capacity is checked each time a customer is added to an existing route. If the capacity of the vehicle is exceeded in any compartment and no other compartment is free, the depot is checked for an available vehicle with a bigger and sufficient capacity for the proposed route. If available, the route is rearranged. If not, the route is not rearranged but ends at the depot and a new route with the depot as a starting point is begun. If the customer's number of demanded products is greater than the number of compartments in the biggest available truck, the customer is served by a minimum of two different trucks. The customer is treated like two separate customers with the same geographical position. The number of routes starting and ending at one depot is never greater than the number of vehicles available at this depot.

The CWS heuristic chooses the next nodes based on their highest savings values. These values are the base of a priority list, which is again randomized. Thus, a different outcome of routes generates thousands of feasible solutions in short computing times and the best solution in a defined maximum time (maxTime) can be chosen.

The result of the biased randomized CWS heuristic is improved with the help of a route cache. As described in chapter 2.3.1 this step is part of the intensive simulation process on elite solutions. During the implementation, a base solution (baseSol) and a defined number of best solutions (bestSol) are defined. 'bestSol' is part of the route cache. At the beginning, the first solution, which is used as a base for the biased randomized CWS heuristic in the routing step, is saved in both.

When the new solutions (newSol) are computed, they are first compared to 'baseSol'. 'baseSol' is replaced if 'newSol' has a better result. In a next step 'newSol' is compared to the best solutions in the route cache. If newSol has lower costs than the most expensive solution in the route cache, the list of best solutions is updated.

After the whole process the solution with the lowest overall costs is returned as 'bestSol'. The outputs after the previously explained steps are the overall costs for inventory and routing, the costs per depot, the routing plan and the refill policy for every depot.

A flowchart of the proposed solving process is displayed in Figure 9.

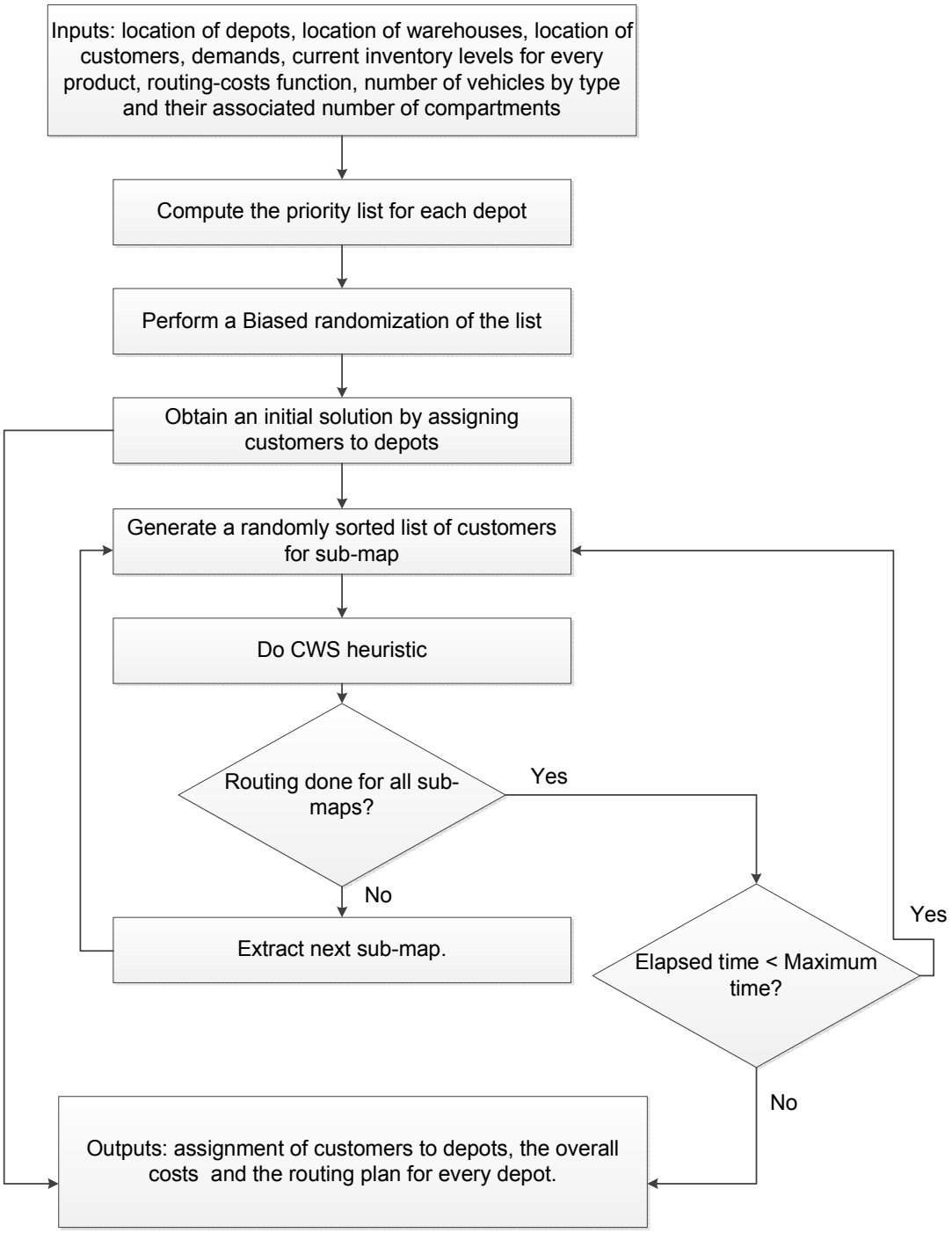

Figure 9 *Flowchart of the Solving Process*

3.4 Formal Description

The single-period Stochastic IRP considered in this work can formally be described as follows. It is defined as a complete undirected graph $G = (V, A)$, where $V = \{V_d, V_c, V_w\}$ is the set of map locations including the depots ($V_d = \{1, .., d\}$), customers ($V_c = \{1, .., i\}$) and warehouses ($V_w = \{1, .., k\}$) and W is the set of edges connecting those locations. In the presented IRP, a number of products $P = \{1, .., n\}$ is distributed. Each customer i has a demand D_{in} to be satisfied for each product. Each depot V_d has to serve several customers for whom the demand

for each individual product is a random variable $D_{in} \geq 0$, following a known probability distribution with $E[D_{in}] = d_{in} > 0$.

Each edge in W has an associated cost $c_{ij} = c_{ji} > 0$, usually computed as the distance between customers i and j (all distances are assumed to be symmetric). For the presented IRP, a solution is a set of routes, in which each route starts at one depot in V_d, has an optional first stop at a warehouse k and connects one or more customers i and ends at the same depot, without exceeding the capacity of the vehicle. Also, routes that start at one depot and end at another depot are not allowed. There are no vehicles based at the warehouses.

The number of vehicles based at each depot (M) may be fixed or unlimited. The former defines a harder problem, since it adds an additional constraint and it is unsure that a feasible solution exists (Chao et al., 1993). The latter simplifies the modeling and solving of the routing step in the IRP (see chapter 3.3). However, in a realistic scenario, the number of available vehicles to satisfy customers' demands is usually fixed.

The number of vehicles at each depot M is a set of small ($M_s = \{1, .., s\}$), medium ($M_m = \{1, .., m\}$) and large vehicles ($M_l = \{1, .., l\}$), each with their own overall capacities $Q_{M_x} > 0$, and number of compartments $N = \{1, .., r\}$ with individual capacities Q_r.

For each depot d, both the current inventory level, $L_{in} \geq 0$, as well as the maximum allowable inventory level, $\widehat{L_{in}} > 0$, are known for every product.

A centralized decision about refill policies for depots' inventory levels requires the decision about the amount of refilled product to be made independently of individually optimized inventory levels. Refill policies are percentages determining the filling level of depots' inventories. This means the decision about each depot's refill policy is made to minimize the amount of demand for every product D_{in}, each depot d holds as inventory and the routing costs it raises. The inventory cost function is as follows:

$$f(D_{in},) = \begin{cases} \lambda_n s_{in} & \text{if } s_{in} \geq 0 \\ 2c_{ij} & \text{if } s_{in} < 0 \end{cases}$$

$\lambda_n \geq 0$ represents the cost of holding a unit of product n in stock at the end of the period that is assumed to be known. s_{in} represents the surplus at the end of the period for every product, i.e., $s_{in} = L_{in} + q_{in} - D_{in}$ for all depots. This calculation assumes that the cost of a stockout of a vehicle on its roundtrip, is the cost of sending a new vehicle from the depot to the customer where the route failure occurred, i.e. a roundtrip.

For every depot a priority lists of potentially eligible customers' locations is computed. For $\forall k \in V_d$, the list associated with k contains all nodes and is sorted according to the distance-based marginal savings, depending on k.

a) First the minimum cost of assigning customer i to $k*$, the best alternative depot to k, is computed, that is $c_i^{k*} = min_{\forall d \in V_d \{k\}}\{c_i^d\}$.
b) Then μ_i^k is computed, the marginal savings in cost associated with assigning customer i to depot d, instead of assigning i to $k*$: $\mu_i^k = c_i^{k*} - c_i^k$.

The marginal savings can be either positive (if k happens to be the "closest" depot to customer i) or negative (otherwise). According to this, nodes that are much closer to one depot k than to the other depots will present higher marginal savings for depot k and, accordingly, will be selected with a higher priority for this depot, occupying the top positions in the sorted priority

list for depot k. On the contrary, nodes located in between two or more depots will present low marginal savings for all depots and they will not become a priority for any depot. The nodes list associated with each depot is sorted following this criterion.

The cost function aims at minimizing all occurring costs. It is stated as follows:

$$min \sum_{1}^{d} set\ up\ costs + routing\ costs + inventory\ costs$$

3.5 Pseudocode

A first version of a pseudocode is presented in this chapter. It is based on the given problem description and the proposed solving process as described in the previous chapters but has some limitations. It is a simplified approach to model the key elements of the proposed solving process. The purpose of this chapter is to explain the proposed pseudocode and its logic. The explanations concerning the limitations content-wise are given in chapter 4.1.1. This pseudocode is later on implemented in Java and thus uses its commands.

The pseudocode has certain inputs: the data provided in the problem description, a maximum time in which the implemented code is searching for feasible solutions and two parameters for a geometrical distribution.

The pseudocode is divided in four sections: inputs/pre-processing, preliminary computations, MS process and outputs. In the first section, displayed in Figure 10, the inputs contain the position of the customers, depots and warehouses, also the number and type of vehicles assigned to each depot and the distance-costs matrix. The customers are implemented with random demands of every available product. The warehouses are implemented with an inventory for every available product with virtually unlimited size. The distance-costs matrix contains the traveling costs between each pair of map locations.

```
-   procedure MultiStart-RichIRP(inputs, maxTime, alpha, beta)
-
-   % INPUTS PRE-PROCESSING
-   % Inputs contain: customers, depots, warehouses, vehicles, and distance-costs ma
    % trix
-   % Note: alpha and beta are parameters of a geometrical distribution for the vari
    % ous performs biased randomizations
-   01 customers <- getCustomers(inputs) % customers with random demands of different
    % products
-   02 depot <- getDepots(inputs) % hold a fixed fleet of different vehicles and a
    % stock per product
-   03 warehouses <- getWarehouses(inputs) % hold a stock per product with virtually
    % unlimited size
-   04 costsMatrix <- getCostsMatrix(inputs) % traveling costs between each pair of
    % map locations
```

Figure 10 *Pseudocode Inputs*

In the second section (preliminary computations), a priority list, based on marginal savings in distance costs, is generated for every depot. After generating a priority list for every existing depot, the list is saved along with all other generated priority lists (Figure 11).

```
-   % PRELIMINARY COMPUTATIONS
-   % For each depot, generate a priority list based on marginal savings in distance
    % costs
-   05 for each {depot in depots} do
-   06   priorityList(depot) <- genPriorityList(customers, depots, costsMatrix)
-   07   priorityLists <- add(priorityList(depot), priorityLists)
-   08 end for
```

Figure 11 *Pseudocode Preliminary Computations*

The third section (MS process) contains the key methods for solving the treated IRP. To save the best computed solution and to compare the new solution with previously computed ones, 'bestSol' is defined. The process to search for new solutions is only running while the maximum time is not exceeded.

Then a balanced customer-depot assignment map is generated. This map is obtained employing a biased randomized round-robin process. Customers are assigned based on the previously computed priority lists for each depot and the remaining capacity surplus of each depot. The depot with the highest overall capacity of products left gets assigned the next customer. The variable 'alpha' is used for the employed distribution of the biased randomization when choosing the next customer. The result after this step is a map with sub-maps. Each sup-map contains a depot with its assigned customers. This step is only finished when all customers are assigned.

In the next step a routing solution is computed for each sub-map. This is done by using a modified version of a biased randomized CWS heuristic to account for multi-product, heterogeneous fleet and stock levels in the depots. The variable 'beta' is used as an input for the used geometrical distribution in the biased randomization. For every computed solution of a sub-map ('subSol') a check for feasibility is performed. Its feasibility depends on the balance of depots' capacities, vehicles' capacities and customers' demands for every product.

If the computed solution of the sub-map is feasible, a local search of routes is conducted by using a memory cache of routes, to further improve the solution. The updated version of the solution is saved as part of the new solution for the complete map.

In a final testing, the costs of the new solution for the complete map are compared to previously computed complete solutions saved as best solutions up to this point of computing solutions. If the new solution's costs are lower than the currently best solution's cost, the best solution is replaced with the new solution and saved as the best. The complete process is described in chapter 3.3.

If the computed solution of the sub-map is not feasible, a new search for a solution is started.

After each completed loop of the described process, the elapsed time is updated (Figure 12).

```
-   % MULTI-START PROCESS
-   09 bestSol <- empty sol with infinite cost
-   10 elapsedTime <- 0
-   11 while {elapsedTime < maxTime} do
-   12   newSol <- empty
-   % 1. Generate a balanced biased randomized customer-depot assignment map using a
    % round-robin process
-   % (i.e., the depot with the highest capacity surplus will select next customer)
-   13   assignMap <- genRandAssignmentMap(customers, depots, priorityLists, alpha)
-   % 2. Generate a routing solution for each sub-map using a modified version of a
    % biased randomized Clarke and Wright Savings Heuristic to account for multi-
    % product, heterogeneous fleet, and stock levels (use smallest possible vehicles
    % first and larger ones as merge processes occur)
-   14   for each {subMap in assignMap} do
-   15     subSol <- biasedRandRichCWS(subMap, warehouses, costsMatrix, beta)
-   16     feasibility <- checkFeasibility(subSol)
-   17     if {feasibility is true} then
-   18       subSol <- localSearch(subSol) % use a memory-cache of routes
-   19       newSol <- add(subSol, newSol)
-   20     else
-   21       feasibility <- false
-   22       exit for
-   23     end if
-   24   end for
-   % 3. If newSol is feasible, update bestSol if appropriate
-   25   if {feasibility is true} and {cost(newSol) < cost(bestSol)} then
-   26     bestSol <- newSol
-   27   end if
-   28   update elapsedTime
-   29 end while
```

Figure 12 *Pseudocode Multi-Start Process*

In the fourth section (outputs) the result of the previously computed best solution is returned and the solving process is ended. The best solution contains the minimal amount of total expected costs of all computed solutions and the related routing plan for every depot. See Figure 13.

```
-
-   % OUTPUTS
-   30 return bestSol
-
-   end procedure
```

Figure 13 *Pseudocode Outputs*

In the next chapter the presented pseudocode is implemented and analyzed in detail.

4 Implementation and Evaluation

4.1 First Implementation

The aim of the first implementation is to get a first working draft of the necessary code. A lot of details are not included in the first version due to the necessity of checking if the basic structure of the idea, as proposed in chapter 3.3, can be implemented. The first implementation does not contain all described steps but establishes some groundwork for further advances. The complexity in the first implementation is reduced for the purpose of obtaining a first working version in a reasonable time frame.

A lot of details that are considered only during actual programming were not considered beforehand, due to the complexity of the problem and the lack of overview. However, it must be noted that when heuristic approaches are developed and implemented, the method is not exact and it is necessary to experiment with different ideas. This is especially the case when the treated problem is new and no benchmarks exist for its solving process, as is the case of the present work.

In chapter 4.1.1 the solving process for the first implementation is described and the differences to the proposed solving process are made clear. In chapter 4.1.2, a manual solution is calculated, a method used by the employees of the company providing the information from which realistic constraints for the treated IRP here are derived (see chapter 3.1). This manual solution is used to verify the first implementation. In chapter 4.1.3 an outlook on further possible improvements of the code is given based on the results of the previous sections. The first implementation was performed in collaboration with Enoc Martinez from the IN3 of the Universitat Oberta de Catalunya, Barcelona.

4.1.1 Solving Process

The first implementation of the code is described in this chapter. The proposed solving process was net of the inventory planning and the inputs are deterministic with fixed customer demands, while all other key elements are implemented. This approach reduces complexity while maintaining a reasonable quality of generated solutions and leads to first test results in an acceptable development time. Also, the adding of the stochastic later is not very time-consuming and thus can be left out for the first implementation.

The beforehand described solving process with the steps assignment, inventory planning and routing is reduced to only the assignment and routing steps. This is due to the exclusion of stochastic values. The main reason for the step of the inventory planning is the variation in customers' demands when vehicles on previously determined routes arrive at the customer. This makes a strategic decision about inventory levels necessary. When the demands are deterministic, as in this first implementation, the consideration of different inventory levels is obsolete, since the customers' demand is fixed and so after the assignment step the necessary inventory is clear for every depot.

The cost function as described in chapter 3.1 cannot be applied here due to the missing inventory calculations. Therefore, in this implementation, it only consists of the set up costs for each tour of the vehicles and the distance-based costs for the total amount travelled of each vehicle. A parameter for the maximum cost ('maxCost') is not introduced at this stage, when calculating assignments of customers to depots and routes, as it isn't given in the problem description. This parameter serves to dismiss calculated solutions as infeasible when it is exceeded. In realistic scenarios this can be added. This is useful in already existing systems which have a given cost and whose solution has to be improved by applying the algorithm.

In the following the assignment and routing are described with special focus on the differences to the proposed solving process in chapter 3.3.

The inputs are as follows: There are three types of vehicles (small, medium and large with a fixed number of compartments with a fixed capacity as described in the problem description) and four products (every customer and every depot has capacities for every product). Also the model is provided with the location of deposits, location of warehouses, location of customers and the routing-cost function.

4.1.1.1 Assignment

In this step the customers are assigned to depots from which they are delivered. The main difference in this first implementation is that instead of an ILS, a MS process is used to obtain maps with customers assigned to depots. In an ILS, a destruction-reconstruction process is performed where an initial solution is randomly partially destroyed and reconstructed. The MS process can be seen as a particular case of an ILS where 100% of the initial solution is destroyed and reconstructed. A well-tuned ILS should perform better by converging faster to a near-optimal solution. However, an ILS structure is also more complex to tune than a MS one. So, when implementing a MS process instead of an ILS, the purpose is to decrease the complexity when obtaining a first version of the solution to reduce the writing time of the code. The MS process already provides a complete initial approach to a complex problem.

As in the proposed solving process, the customers are assigned to depots with a capacitated round-robin process. It is assumed that the inventory levels are set at 100%. While assigning customers to the depots, the depot with the highest capacities left is chosen to be assigned to the next customer. The customers are chosen from a priority list based on marginal savings and with the help of biased randomization. The distances between all customers, depots and warehouses are computed with coordinates and the Euclidean distance.

As described in the proposed solving process, an element is embedded in the code that introduces randomness, called a random number generator (RNG). This is used for the biased randomization in the assignment and routing steps of the algorithm. This RNG uses the static Java method `Math.random` and in this implementation the method produces distributions, returning random numbers from 0 to n-1 with a given n, defining the maximum number of characteristics.

In order to check if a map is valid during the assignment process, the overall customer demand has to be smaller than the sum of the vehicles' capacities assigned to the depot. The number and type of vehicles assigned to one depot stays the same during the whole process.

When the vehicles' capacities are sufficient but the depots' capacities are not, the assignment is still valid. But then it is assumed that one vehicle has to start from the warehouse, which implies a penalty cost for the additional distance covered by the vehicle. This cost is set at 100 monetary units. The distance-based variable cost is set at 0.5 monetary units per unit of length. These are just initial values and can be adapted to values used in a real scenario any time later. The vehicles are required to arrive empty at the warehouse. Since the final code is planned to have a fluctuating customer demand, the only option is to let the vehicles start at the warehouse when a stop there is necessary. The warehouses in the IRP treated in the present work have the condition that the arriving vehicles have to be empty. If customers' demands vary, this condition can only be fulfilled when vehicles stop at the warehouse at the beginning of their routes, since the exact fill level of the vehicles can never be calculated previously when dealing with stochastic values. The warehouses in this scenario have virtually unlimited stock and thus are not included when calculating the capacities.

The feasibility of the assignment is only based on the balance of the capacities of depots and vehicles and the customers' demands. The distance between depots, customers and warehouses is not considered in a realistic context at this stage. That is, time windows are not considered, so vehicles can cover a distance that may not be manageable in real companies. This would mean another constraint and make the problem too complex at this stage of the present work.

The result of the assignment step is shown in Figure 14.

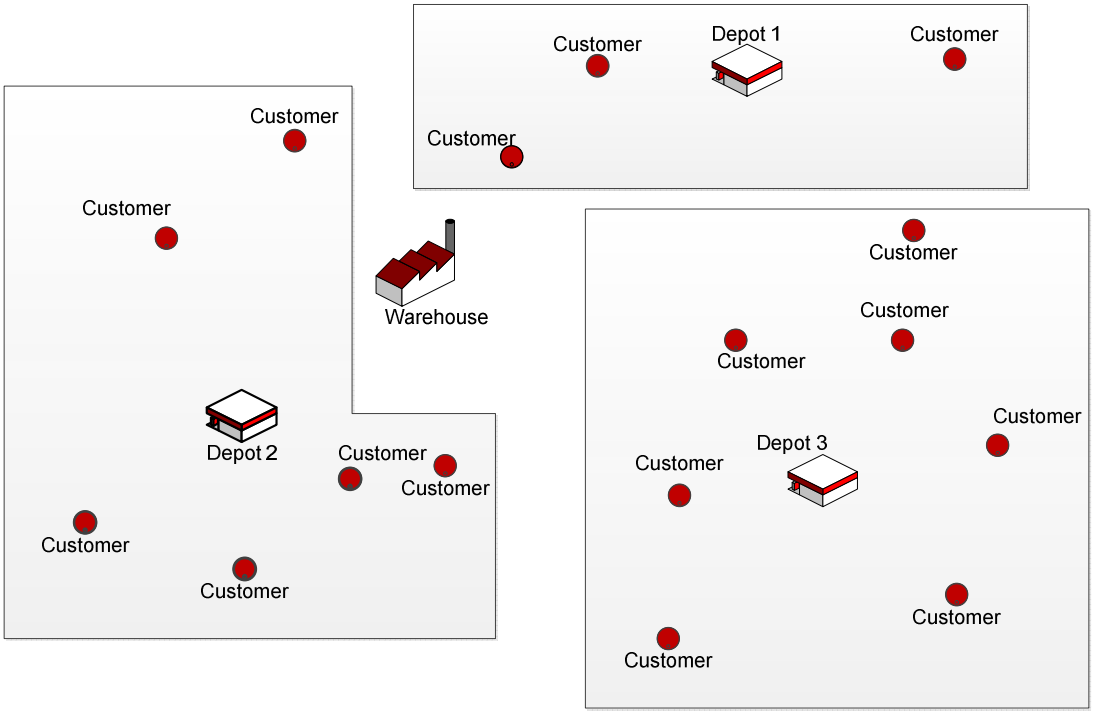

Figure 14 *Assignment of Customers to Depots*

All customers are assigned to a depot and the warehouse can be accessed from every depot by the vehicles assigned to each depot.

4.1.1.2 Routing

As proposed in chapter 3.3, an initial solution is improved using a randomized CWS heuristic by choosing the vehicles with the biggest surplus of overall capacity. The aim is to generate the highest amount of savings in distance that is possible.

The initial solution is obtained by employing an, at first, unlimited number of the smallest vehicles possible. They are used to form routes directly to and from the depots to every customer assigned to it. This process results in a number of routes which is equal to the number of customers served. This initial solution is based on the assumption that the number of compartments is always sufficient for the number of products requested by one customer in this first implementation. This assumption is valid even if the overall number of products exceeds the number of compartments in the biggest vehicle. This is due to the fact that the problem treated in the present work is based on a realistic IRP in which the products are mainly oil and gas, i.e., the customers (who are mostly private individuals) will most likely not request a broad variety of the products offered by the company.

When merging the routes during the randomized CWS heuristic, all the vehicles available at one depot are ideally used to generate the highest savings possible. So when merging two nodes, the new route with the highest savings value chooses an available vehicle with the highest surplus of capacity. This leads to assigning the biggest vehicles to the routes with the highest savings. By applying this logic to the code, the biggest vehicle is most likely to be chosen first and the smallest vehicle last. For this first implementation there is no other criterion apart from availability linking vehicles and customers. But since the merging is done with the help of biased randomization, the matching of saving values and size of vehicles is most expected to have a reasonably good result. This is caused by the functionality of biased randomization (see chapter 2.3.2). By biased randomizing the decisions about choosing a vehicle for a tour, alternative scenarios are considered and the solution with the best result is returned as an output when running the code. When implementing metaheuristic approaches, there is no clearly defined way for finding the best solution. So it has to be experimented with different ways of solving a given problem and thus finding the best solution in a given time.

If the biased randomized CWS heuristic is the only tool used to solve the routing plan, problems can arise because the code picks new routes based on saving values to which probabilities with the best outcome are assigned. Even if e.g., the highest saving value on top of the generated priority list is, in total numbers, already much better than the saving value on the 2^{nd} position, then the 2^{nd} position still is eventually picked based on a certain probability and the result can be expected to be much worse due to a much lower savings value. This circumstance can be corrected using a route cache, where better routes than new ones being generated are saved and a comparison between the two will lead to a termination of the routing in progress. This saves computing time and improves the results significantly.

When merging the routes, not just the overall capacity of assigned vehicles is considered but also the capacities for each individual product and the compartments' sizes of the used vehicle. In doing so the biggest compartment is matched with the product with the biggest amount to be transported. This follows again the principle of matching the highest demand with the highest surplus of capacities as used before.

The routes including a stop at a warehouse first, are treated with a special approach. The selection about which node is approached first after the warehouse is performed again. This is performed by choosing the node with the highest marginal savings from the viewpoint of the warehouse. This is a first approach that can be extended to a biased randomized selection of first nodes to approach after the warehouse. But it serves the purpose for the first implementation.

The routing process is visualized in Figure 15.

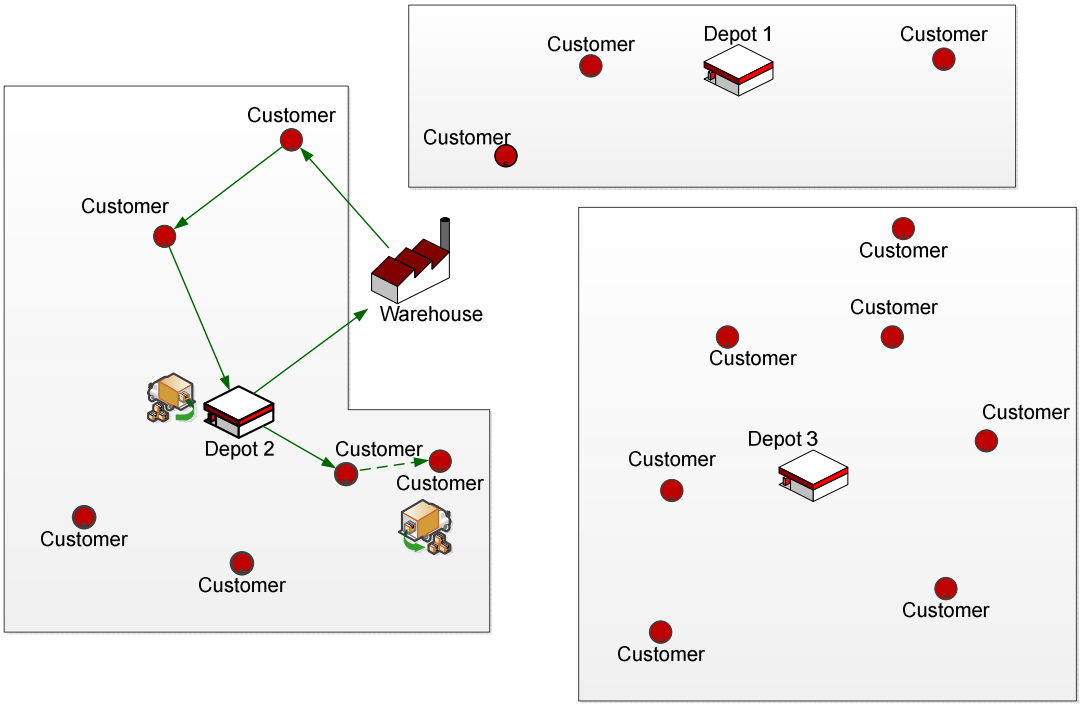

***Figure 15** Routing Process*

Time windows are again not considered in this first implementation due to the effort in programming that comes with such an additional complexity in an IRP.

In the next section a first evaluation is conducted of the described implementation.

4.1.2 Case Study

When solving a given problem with heuristics, the solving process is a combination of various individual elements. The quality of the elements is based on their outcome in previous testing or knowledge obtained from previous research with given results. Accordingly, when dealing with a heuristic solving process as in the present work, the chosen methodologies are only one possible solution. In order to determine the benefit of the first implementation, in this chapter a comparison is conducted between the implemented first code and a manual solution 'by eye' that is practiced in the company providing realistic information on IRPs. This provides an impression whether the first implementation in Java is superior.

In the company, an experienced employee, familiar with the company's system and capacities, determines the assignment of customers to depots and the routing of vehicles without the assistance of computers or any other methods or techniques. The employee considers the

customers' locations and demands on a map and screens the available capacities of deposits and vehicles. Afterwards assignment and routing with certain constraints (see chapter 3.2) are determined, based on the proximity of map locations. This solving method is used in this chapter for comparison to the developed implementation to evaluate the advantage in cost and time management efficiency such a program can represent.

In the following chapters various tests are conducted with increasing problem size. The algorithm is implemented and run as a Java application on a standard personal computer (Intel Core 2 Duo processor, at 2.0 GHz with 4 GB RAM).

4.1.2.1 First series of test

The first manual test is based on random inputs as used in the first implementation. The used values are only to a small proportion related to the described company's problem since exact locations and demands of their customers are not provided. The focus is merely on the functionality of the code and the solving process, therefore the importance of the inputs is negligible as long as they fulfill the main constraints as described in chapter 3.2. The inputs are listed as follows.

- The number of customers is set at i = 20.
- The number of depots is set at d = 3.
- The number of warehouses is set at k = 1.
- The fixed set up costs for sending one vehicle on one route amounts to 100 monetary units.
- The variable distance-based costs are set at 0.5 monetary units per unit of length.
- The numbers of products is set at n = 2.

The customers' IDs are used to identify their unique location and assigned demands, the customers' x and y coordinates and the demand for both products are displayed in Annex 1.

The depots are also described with their ID, x- and y-coordinate and the capacities for every product. The values are displayed in Table 1.

Depot ID	X-Coordinate	Y-Coordinate	Capacity Product 1	Capacity Product 2
21	20	20	1,000	300
22	30	40	1,000	100
23	50	30	1,000	100

***Table 1** Depots Inputs Test 1*

The following vehicles are assigned to the depots.

- Depot 21 and 22: one vehicle with two compartments and the capacities 50 and 50, two vehicles with two compartments and the capacities 200 and 200.
- Depot 23: two vehicles with two compartments and the capacities 100 and 100.

Finally the warehouse is described with its ID and the x- and y-coordinate in Table 2.

Warehouse ID	X-Coordinate	Y-Coordinate
24	70	70

Table 2 *Warehouse Inputs Test 1*

When calculating the manual solution, the constraints especially include the following ones and are valid for all following implementations and case studies:

- All customers' demands need to be satisfied 100% every time.
- All routes start and end at the same depot.
- The distances between map locations are computed using the Euclidean distance.
- Only one iteration is performed when solving the problem manually. No improvement procedures are performed.

When computing the manual solution, no methods, as used in the implementation, are applied, in order to generate the same solving process as the employee uses. The only factors determining the assignment and routing, are the proximity of locations and sufficient capacities of depots and vehicles. The necessary time to compute the manual solution is set as a maximum time when running the first implementation and comparing the two results of the best solutions.

For the completion of the solving process, a coordinate system with the locations of customers, depots and the warehouse is drawn. Then the assignment step is performed, considering the maximum capacities of depots and vehicles and customers' demands. The minimum number of required vehicles is determined as well as if the necessity of the warehouse as a first stop of a route arises. When no stop is necessary, the performed routes can be done in either direction.

The result of the manual solving process is displayed in Table 3.

	Depot 21	Depot 22	Depot 23
Assignment	4, 13, 14, 15, 17, 18, 19	1, 5, 6, 7, 8, 12	2, 3, 9, 10, 11, 16, 20
Routing	19-13-14-18 -4-17-15	8-7-6-1-5-12	3-2-11-16-9-20-10
Necessity of stop at WH	No	Yes	Yes
Distance covered in units of length	96,13	179,91	129,97
Variable costs in monetary units	0,5*96,13=48,065	0,5*179,91=89,955	0,5*129,97=64,985
No. of routes/necessary vehicles	1	1	1
Set-up costs in monetary units	1*100=100	1*100=100	1*100=100

Table 3 *Result of the Manual Solution Test 1*

The necessary time of the assignment and routing for solving the given problem was 6 minutes and 31 seconds. This is a tolerable time in a real-life scenario when an employee has to generate a feasible solution in a short period of time every day. The total costs of this solution add up to 503.01 monetary units.

The same input and time frame (6min 31s) is used for computing a feasible solution by running the code. 988,376 solutions were computed in this time. The result of the best solution is shown in the following Table 4.

	Depot 21	Depot 22	Depot 23
Assignment	1, 2, 4, 6, 7, 8, 10, 12, 14, 15, 16, 17, 18	5, 11	3, 9, 13, 19, 20
Routing	Vehicle 1: 4 Vehicle 2: 15-10-16-2-8-7-1-6-14-18-12-17	Vehicle 1: 5 Vehicle 2: 11	20-9-3-13-19
Necessity of stop at WH	No	No	No
Distance covered in units of length	200.8	52.36	151.38
Variable costs in monetary units	0.5*200.8=100.4	0.5*52.36=26.18	0.5*151.38=75.69
No. of routes/necessary vehicles	2	2	1
Set-up costs in monetary units	2*100=200	2*100=200	1*100=100

Table 4 *Result of the Computational Solution Test 1*

This concludes total costs of 702.27 monetary units. The graphical result of the computation is shown in Figure 16.

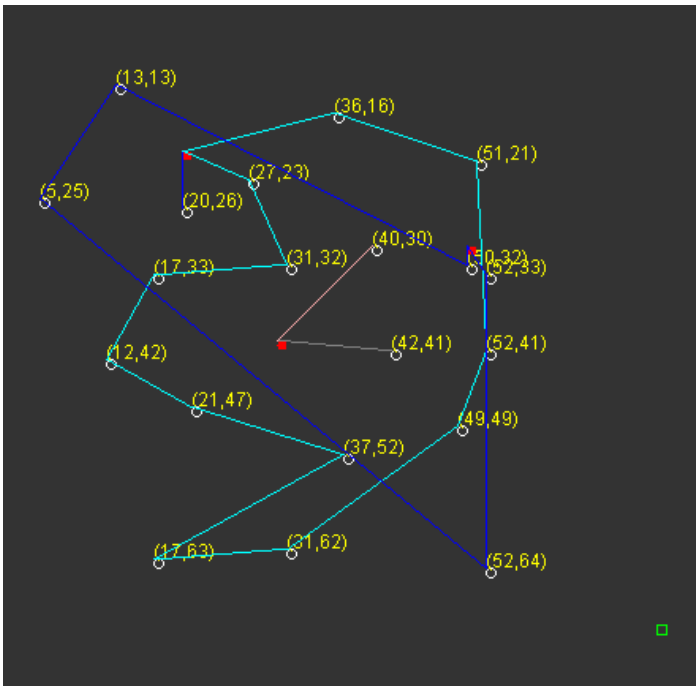

Figure 16 *Graphical Computational Solution Test 1*

The depots are shown in red, the warehouse in the lower right corner in green. The customers are displayed in yellow and the routes connecting the nodes have a different color for every used vehicle.

Comparing the manual and the computational solution, the costs of the computational result are 39.61% higher than the manual solution.

This result leads to the conclusion that the manual solving process is superior to the computational one, when applied to a problem as above with low complexity. The assumption is though that the more complex a problem gets, the more difficult it is to calculate a solution superior to the computational results to the same problem. Thus in the next chapter the problem size is increased and the same comparison is conducted with a larger input size.

4.1.2.2 Second series of test

In the second test the inputs are modified so that they are building a more complex problem to solve. This serves the purpose of proving that the more complex a problem gets, the more difficult it is for a manual solution to be superior to the computational result. Since in the case of IRPs the complexity is very high, computational results provide a way of solving those problems efficiently. In order to make the problem solvable in a manual way, the input problem is still not too extensive.

In Annex 2 the customer inputs for this series of test are displayed. They consist of a number of customers i = 100 and the number of demanded products n = 3. In Table 5 the depot inputs are displayed.

Depot ID	X-Coordinate	Y-Coordinate	Capacity Product 1	Capacity Product 2	Capacity Product 3
101	20	20	1,000	300	200
102	30	40	1,000	100	100
103	50	30	1,000	100	100

Table 5 *Depots Inputs Test 2*

The following vehicles are part of the company's fleet:

- Small vehicle: 4,000L overall capacity with two compartments (50%/50%).
- Medium vehicle: 9,000L overall capacity and three compartments (20%/35%/45%).
- Large vehicle: 30,000L overall capacity and three compartments (20%/35%/45%).

The fixed assignment of vehicles to depots is as follows:

- Depot 101: 2 small, 1 medium, 1 large vehicle
- Depot 102: 1 medium vehicle
- Depot 103: 1 small, 1 medium vehicle

This is the first step of testing the treated IRP in a manual test. The overall capacity still is much higher than the demand while the number of customers is increased. Thus the complexity of the problem is increased gradually. Only the medium and large vehicles are allowed to stop at the warehouse.

Finally the warehouse is described with its ID and the x- and y-coordinate in Table 6.

Warehouse ID	X-Coordinate	Y-Coordinate
104	70	70

Table 6 *Warehouse Inputs Test 2*

The same solving process and the same constraints are applied to this given problem as described in chapter 4.1.2.1.

The result of the manual solving process is displayed in Table 7.

	Depot 101	Depot 102	Depot 103
Assignment	4, 13, 14, 15, 17, 18, 19, 26, 33, 34, 38, 39, 44, 52, 53, 54, 58, 59, 67, 73, 74, 77, 78, 82, 84, 85, 90, 91, 92, 95, 96, 97, 99	1, 3, 6, 7, 8, 11, 12, 24, 27, 28, 32, 35, 37, 41, 45, 46, 47, 48, 55. 57, 61, 64,65, 66, 68, 72, 79, 83, 86, 88	2, 5, 9, 10, 16, 20, 21, 22, 23, 25, 29, 30, 31, 36, 40, 42, 43, 49, 50, 51, 56, 60, 62, 63, 69, 70, 71, 75, 76, 80, 81, 87, 89, 93, 94, 98, 100
Routing	38-4-34-97-58-19-54-73-13-33-53-91-14-74-96-82-85-18/67-39-92-59-78-44-26-52-17-90-15-99-77-95-84	32-72-37-86-6-64-27-79-3-7-47-66-46-24-28-8-61-65-83-88-1-41-45-11-35-55-12-68-48-57	63-20-9-30-76-94-87-42-29-81-22-60-2-23-80-50-16-98-70-51-89-21-5-93-75-25-71-31-100-40-36-49-10-56-43-62
Necessity of stop at WH	No	No	No
Distance covered in units of length	170.96	162.78	165.78
Variable costs in monetary units	0.5*170.96=85.48	0.5*162.78=81.39	0.5*165.78=83.28
No. of routes/necessary vehicles	1	1	1
Set-up costs in monetary units	1*100=100	1*100=100	1*100=100

Table 7 *Result of the Manual Solution Test 2*

The necessary time of the assignment and routing for solving the given problem was 21 minutes and 12 seconds. This is a tolerable time in a real-life scenario when an employee has to generate a feasible solution in a short period of time every day.

The total costs of this solution add up to 549.76 monetary units.

The same input and time frame (21min 12s) is used for computing a feasible solution by running the code. 74,491 solutions were computed in this time. The results of the computational solving are displayed in Table 8 and in Figure 17.

	Depot 101	Depot 102	Depot 103
Assignment	2, 6, 17, 18, 22, 26, 33, 34, 35, 38, 41, 42, 45, 53, 56, 60, 61, 67, 79, 82, 93	1, 7, 11, 14, 17, 23, 25, 26, 29, 35, 37, 38, 40, 45, 47, 48, 49, 55, 56, 57, 60, 61, 63, 66, 69, 77, 82, 85, 87, 89, 90, 95	2, 9, 20, 22, 34, 37, 46, 54, 55, 57, 61, 64, 84, 100

Table 8 *Result of the Computational Solution Test 2*

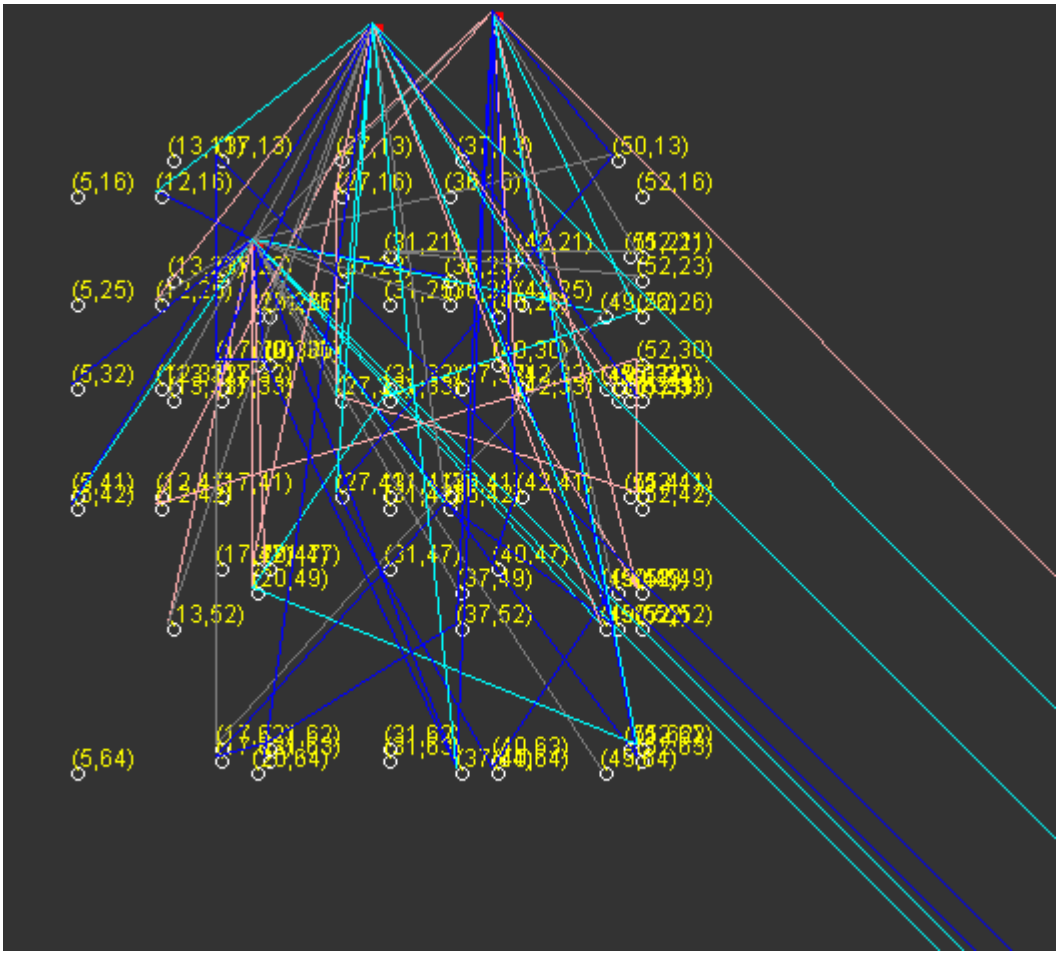

Figure 17 *Graphical Computational Solution Test 2*

The results as displayed suggest a malfunction of the implemented code. Since the first test was already conducted with this version of the code, the second test is terminated here for a lack of possible comparisons. For improved results and for a following correct verification and validation process the code is successfully improved and described in chapter 4.2.

4.1.3 Outlook

Based on the first implementation and the proposed solving process an outlook is given in this chapter on possibilities to improve the first version of the implemented code.

The differences of the first implementation to the proposed solving process are discussed in chapter 4.1.1. In the current chapter an analysis is conducted of observations made during the first implementation process, with a special focus on details that were not detected as relevant before.

During the implementation it is assumed that the number of vehicles never exceeds the number of routes and also that the number of compartments is always sufficient for transporting all products in demand. It is also assumed that one customer is only visited by one vehicle per period. When aiming at implementing a general approach to a problem as in the present work, it is not expedient. While the code allows the modification of the number of compartments, products, number of vehicles etc., it still includes the assumption as stated above. This can lead to unfeasible solutions when not all customers' demands can be served. In order to avoid this, an approach is to allow one vehicle for driving more than one route. When customers order more products than there are compartments in the biggest available vehicle, the order can be split and one customer can be treated as two customers with the exact same location. This way, at least one other vehicle can deliver the rest of the demanded products. This approach is also useful when one customer orders more products than there is capacity in the biggest available vehicle. When stochastic demands are used as inputs, the change in customers' demands can lead to route failures as described in chapter 3.3.

When considering the possibility of various vehicles visiting one customer and one vehicle being able to drive several routes, time windows for deliveries and a maximum time or maximum distance covered in all routes become more relevant. Time and covered distance were not considered in the first implementation due to too much complexity. But considering realistic inventory routing systems, time and distance are an important influence on costs and fulfilled demands e.g., drivers of vehicles cannot exceed a certain maximum driving time. Additionally, based on the current system only linear distances are calculated between map locations, not actual distances that would arise on roads connecting the locations. Realistic distances would be the route travelled on the roads between nodes. When implementing a system to compute distances that can lead to route failure, when exceeding a defined maximum distance, exact distances between nodes become more relevant. This is possible by implementing a tool using e.g., Google maps, calculating the distances between two exact locations on a map. This way time and distance are easier to measure and can be used as an indicator for route failures.

During the implementation and testing of the routing process, a few issues became apparent that are listed below:

- Crossing of routes, e.g. as displayed in Figure 16, which may lead to inferior solutions.
- One vehicle for one customer only even though the capacity would allow for much more and thus spending a lot on set up costs instead of checking if the customer could be served by another vehicle on an already existing route.

- There is no implemented functionality which aims at designing routing “circles”, instead the routing is done randomly and thus enables a going forth and back between nodes which in realistic scenarios can lead to inefficient solutions.

Further observations are listed below:

- When using the capacitated round-robin process to assign customers to depots, only the overall capacity of products is considered but not individual products when deciding which depot gets picked next for the assignment. This can lead to an assignment of customers without sufficient stock at the depot to serve them. A solution could be to consider every product individually during the assignment step.
- When routes include a trip to the warehouse, the choosing of which customer is the first stop after the warehouse could be biased randomized. In the first implementation the first customer is chosen based solely on the lowest distance to the warehouse which can lead to suboptimal results.
- The cost function should be fixed so it can work correctly. It also can be amended by including arising costs for additionally purchased products, depending on the purchased amount, when including the warehouse in a trip. Also, when implementing a heterogeneous fleet, different variable costs can be associated with different vehicles e.g., depending on their size and features.

These observations only serve as ideas on how to further improve the implementation and to make the solving process more realistic. It is most likely that not all proposed changes can be implemented due to the already high complexity of the IRP and the solving process. They serve as impulses for further improvement.

4.2 Second Implementation

The second implementation was done based on the previously described one and extended in collaboration with Gabriel Alemany, student at the Universitat Oberta de Catalunya, Barcelona. The basic structure of the processes and the single steps are not changed but they are refined and include several adaptations as planned in the outlook in the previous chapter. In the following sections, the solving process is described (4.2.1) and a case study is conducted (4.2.2) based on the second implementation. To avoid redundancies, not the whole structure of the solving process is described again but the changes conducted compared to the previous implementation in chapter 4.1.

4.2.1 Solving Process

In the second implementation the focus is on refining the routing procedures, eliminating errors and to reach a positive validation. For this purpose and to experiment, the input values are still deterministic and the inventory levels of the depots at 100%.

The most extensive change was conducted in the implementation of the routing. In the first implementation, a lot of routes were crossing each other, supplying customers further away

from the depot than other customers and thus destroying the logic behind the heuristic procedure of finding feasible routes. Improving routes in the first implementation was only a small part of the algorithm. It was done by comparing three consecutive edges forming a route in different combinations. This procedure is displayed exemplary in Figure 18.

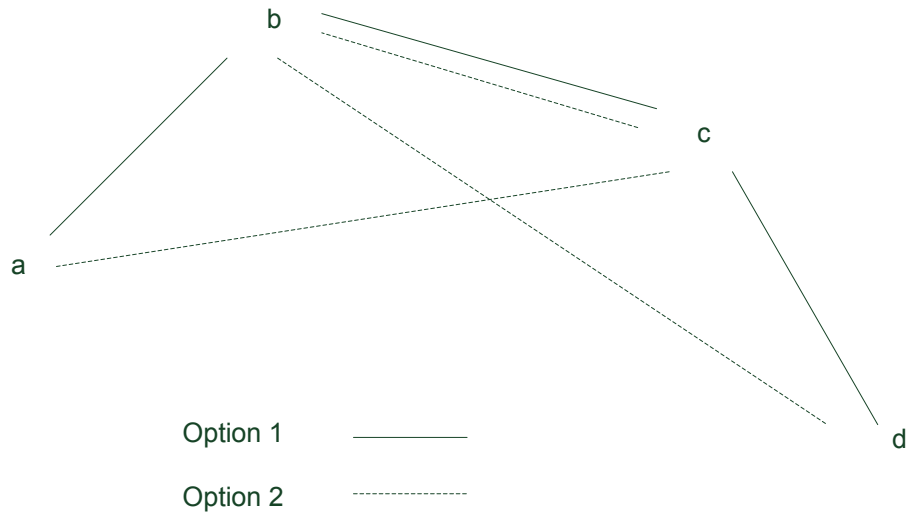

Figure 18 *Routing Improvement Implementation 1 (based on Croes 1958)*

When forming routes, the algorithm of the first implementation would consider different routing options and choose the one with the lowest overall costs. But the algorithm would do this only for three consecutive segments of a routing option.

In the second implementation, the improvement of the routing is performed using a 2-opt algorithm. In optimization, 2-opt is a local search algorithm without perturbations first proposed by Croes (1958) for solving the TSP. It has the same functionality as the approach described above but extends the check over the three consecutive road segments for complete routes and thus removes knots of routes and minimizes routing costs significantly. In the first implementation, every segment (the route between map locations) was treated as just one single segment. In the second implementation each segment consists of the number of products that are transported on this segment. So it is a set of segments, or sub-routes, equal to the total number of products transported on the route. In this implementation it is possible to have customers being visited by more than one vehicle. This solves the in the previous implementation existing problem in case the number of products is bigger than the number of compartments in the biggest available vehicle and also allows for more economical routing solutions. So when creating sub-routes, the algorithm also creates sub-costumers, each with a demand of exactly one product. So one customer will consist of as many sub-customers as the customer requests products. Vehicles can pursue one sub-route and deliver any amount of product they have loaded. If a parallel sub-route is similar to the first one and the vehicle has loaded the product in demand, it can supply it as well. As every sub-customer only demands one product, the route between two sub-customers belonging to the same customer is measured with a distance of 0 units of length for the vehicle.

During the routing, the depot is maintained as the starting and ending point of a route. When a route includes the stop at a warehouse, which has to be the first stop of a route, then the algorithm disallows to break the link between the depot and the warehouse. At the end of the routing the algorithm checks if the warehouse is at the beginning or end of a route. If it is located at the end, the whole route is reversed.

In this work a particular case of a problem with only one warehouse is solved. But the work aims at a general algorithm. Thus, when optimizing the routing with several warehouses, the picking of the warehouse to be included in a route is as follows: when the routes are already formed, the warehouse nearest to the center of the route is added as a first stop to the route. Finally the route is again optimized and the route is changed as needed.

When a stop at the warehouse is necessary, there is no penalty cost but instead the additional distance cost for driving to the warehouse and afterwards to the first node of the following route is computed and added to the total costs.

In the first implementation, when using the capacitated round-robin process to assign customers to depots, only the overall capacity of products is considered but not individual products when deciding which depot gets picked next for the assignment. This approach is extended to a two-phase round-robin process.

In the first phase, a basic round-robin process is applied to the depots. For each of the depots, a priority list of customers is computed and the depots pick one customer each round. This process is continued until one depot runs out of capacities. Then the second phase begins and continues to assign the remaining customers to the depots with remaining capacities.

During the second phase, a capacitated round-robin process is employed as described in chapter 3.3. The depot with the highest surplus of transport capacities chooses the next customer from its priority list for high savings. This way the advantages of an evenly distributed number of customers and assigning more customers to depots with a higher capacity are exploited. This approach creates a map with comparably balanced sub-maps. Thus not only the depots' capacities are considered but also the distribution of customers' locations.

In the next chapter a case study is conducted evaluating the second implementation.

4.2.2 Case Study

The inputs for this case study were created randomly to a great extent due to unknown customer locations and demands in the given problem. The number of depots, warehouses, vehicle types and associated vehicle compartments were adopted from the original problem.

When the demands were created, for practical purposes the only constraint was they would not exceed the combined vehicles' capacities of all existing depots.

The inputs are listed as follows.

- The number of customers is set at i = 100.
- The number of depots is set at d = 3.
- The number of warehouses is set at k = 1.
- The fixed set up costs for sending one vehicle on one route amount to 100 monetary units.
- The variable distance-based costs are set at 0.5 monetary units per unit of length.

- The numbers of products is set at n = 4.

In Annex 3 the complete customer inputs for this series of test are displayed. An additional constraint includes that it can be determined for vehicles whether they are allowed to drive to the warehouse or not. In the following Table 9 the depot inputs are displayed.

Depot ID	X-Coordinate	Y-Coordinate	Capacity Product 1	Capacity Product 2	Capacity Product 3	Capacity Product 4
101	29	21	74,742	81,408	86,250	72,762
102	154	19	70,836	62,394	71,706	63,066
103	64	62	81,924	82,422	68,358	66,804

Table 9 *Depot Inputs Test 3*

The following vehicles are part of the company's fleet:

- Small vehicle: 4,000L overall capacity with two compartments (50%/50%).
- Medium vehicle: 9,000L overall capacity and three compartments (20%/35%/45%).
- Large vehicle: 30,000L overall capacity and three compartments (20%/35%/45%).

The fixed assignment of vehicles to depots is as follows:

- Depot 101: 7 small, 7 medium vehicles
- Depot 102: 7 small, 7 medium vehicles
- Depot 103: 7 small, 5 large vehicles

The warehouse is described with its ID and the x- and y-coordinate in Table 10.

Warehouse ID	X-Coordinate	Y-Coordinate
104	60	36

Table 10 *Warehouse Inputs Test 3*

Due to the case study's complexity, for the manual solution the time frame was set at one hour to simulate a realistic company environment where an employee can prepare an assignment and routing plan for the day.

The solving of the given problem was started randomly with depot 102. After the expiration of one hour, the assignment was conducted but the routing and the capacity check for the vehicles delivering the customers was only performed for only about four vehicles of one depot. This is owed to the long times for checking the capacities of available vehicles, their compartments and the customers' demands. Not even a quarter of all customers was evenly supplied, almost none were completely supplied. The results after one hour are shown in Table 11.

	Depot 101	Depot 102	Depot 103
Assignment	1, 2, 3, 4, 6, 7, 13, 14, 15, 16, 17, 18, 20, 21, 22, 24, 25, 26, 27, 28, 29, 30, 31, 34, 35, 38, 39, 43, 44, 45, 46, 52, 54	5, 9, 10, 11, 12, 19, 23, 33, 37, 42, 49, 50, 51, 58, 65, 67, 77, 78, 85, 89, 90, 97	32, 36, 40, 41, 47, 48, 53, 55, 56, 57, 59, 60, 61, 62, 63, 64, 66, 68, 69, 70, 71, 72, 73, 74, 75, 76, 79, 80,81, 82, 83, 84, 86, 87, 88, 91, 92, 93, 94, 95, 96, 98, 99, 100
Routing	N.A.	Vehicle 1, Type small: 9-10-5-12-19 Vehicle 2, Type medium: 37-50-49-65-67-77-78-89-90 Vehicle 3, Type medium: 11-12-23-33-42-51-58-85 Vehicle 4, Type medium: 23-42-51-58-85-97-90-89-78 *rest remains	N.A.
Necessity of stop at WH	N.A.	No *rest remains	N.A.
Distance covered in units of length	N.A.	510.72 *rest remains	N.A.
Variable costs in monetary units	N.A.	510.72*0.5=255.36 *rest remains	N.A.
No. of routes/necessary vehicles	N.A.	4 *rest remains	N.A.
Set-up costs in monetary units	N.A.	4*100=400 *rest remains	N.A.

Table 11 *Result of the Manual Solution Test 3*

57,514 solutions were computed in the given time frame of one hour. The results of the computational solving are displayed in Table 12 and in Figure 19. For the complete output of the program the reader is referred to Annex 4. It includes the total costs, the assignment and the routing but is summarized with the information given in Table 12.

	Depot 101	Depot 102	Depot 103
Assignment	1, 2, 6, 7, 8, 13, 14, 20, 21, 22, 24, 25, 27, 28, 29, 34, 38, 39, 43, 44, 45, 52, 54, 59, 68	4, 5, 9, 10, 11, 12, 16, 18, 19, 23, 32, 33, 37, 42, 49, 50, 51, 58, 65, 67, 77, 78, 85, 89, 90, 97	3, 4, 15, 16, 17, 18, 26, 32, 36, 40, 41, 46, 47, 48, 53, 55, 56, 57, 60, 61, 62, 63, 64, 66, 69, 70, 71, 72, 73, 74, 75, 76, 79, 80, 81, 82, 83, 84, 86, 88, 91, 92, 93, 94, 95, 96, 98, 99, 100
Routing	Vehicle 1: 35-45-28-29-25-14-22-2-1	Vehicle 1: 11-5-10-9-18-16-32	Vehicle 1: 84-100-88-96-76-73-72-63-62-57-47-36-48-41-17-3-15-31-30-56
	Vehicle 2: 21-22-2-1-8	Vehicle 2: 23-42-33-32-16-4-9	
	Vehicle 3: 7-6-13-24-27	Vehicle 3: 19-12-23-51-37	Vehicle 2: 75-100-88-96-76-73-63-64-41-40-48-47-26-16-3-15-31-30-46-60-80-79-91-86-92-81-56-61
	Vehicle 4: 7-20-6-13-24-34-27	Vehicle 4: 37-49-65-77-89-90-67	
	Vehicle 5: 35-44-39-25	Vehicle 5: 51-58-85-97-78	Vehicle 3: 62-53-57-61-66-55-60-70-69-74-98-93-94-82
	Vehicle 6: 38-59-68-54-52-43-34	Vehicle 6: 11-5-10-9	Vehicle 4: 95-87-94-82-71
	Vehicle 7: 45-59-68-52-43-38	Vehicle 7: 33-58-85-89-77-65-49-50	Vehicle 5: 83-87-99-81-93-92-86-91-79-80-69-70-60-55-66
Necessity of stop at WH	No	No	No
Distance covered in units of length	N.A.	N.A.	N.A.
Variable costs in monetary units	123.82 + 91.91 + 55.65 + 70.02 + 75.92 + 109.18 + 101.81 = 628.31	177.39 +.205.67 + 80.86 + 115.69 + 125.73 + 41.85 + 133.18 = 885.37	244.02 + 354.19 + 180.48 + 63.53 + 157.45 = 999.67
No. of routes/necessary vehicles	7	7	5
Set-up costs in monetary units	N.A.	N.A.	N.A.

Table 12 *Result of the Computational Solution Test 3*

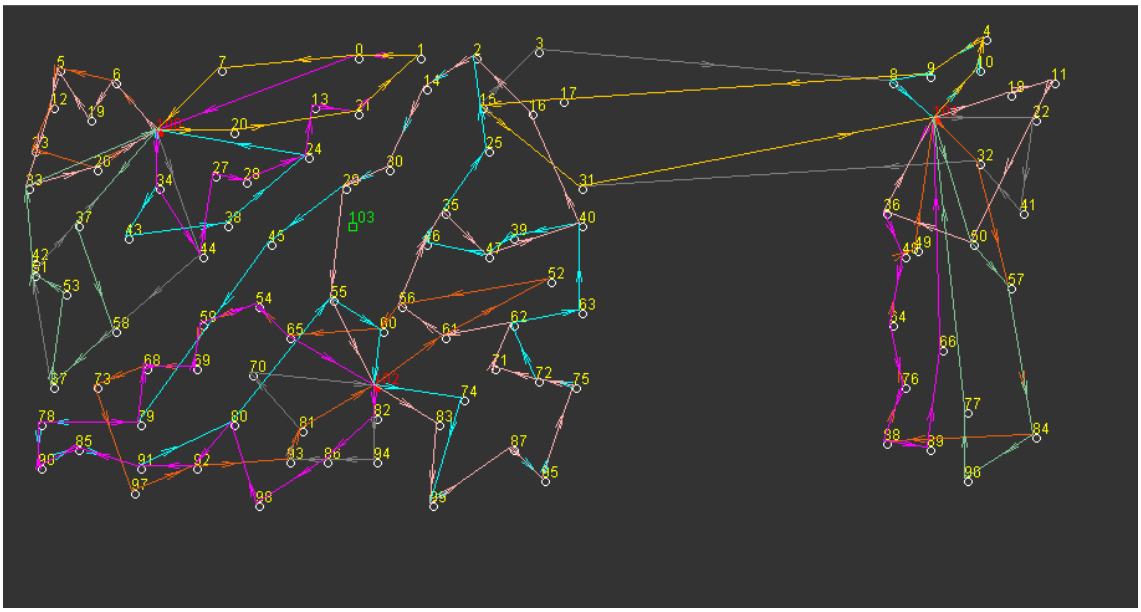

Figure 19 *Graphical Computational Results Test 3*

The depots are shown in red, the warehouse in the lower right corner in green. The customers are displayed in yellow and the routes connecting the nodes have a different color for every used vehicle. The arrows included in the routes imply the vehicles' travelling direction.

The total costs amount to 2,508.37 monetary units as an output provided by the code. A manual review however led to the specification of costs as displayed in Table 13.

	Depot 101	Depot 102	Depot 103
Distance covered in units of length	531.72	890.90	998,53
Variable costs in monetary units	531.72*0,5 = 265,86	890.90*0.5 = 445.45	998.53*0.5 = 499.27
No. of routes/ necessary vehicles	7	7	5
Set-up costs in monetary units	7*100 = 700	7*100 = 700	5*100 = 500

Table 13 *Manual Review of Computational Results Test 3*

As demonstrated, the costs differ from the costs provided in Table 12. The overall costs in the review amount to 3,084.42 when considering the set up costs and costs per km as indicated above.

Further comparison of the manual and the computational solution shows that the assignment of the customers to depots in the computational solution was based more on the overall capacities of vehicles assigned to the individual depot. In the manual solution, the reason for assigning a customer was rather the minimal distance to a depot. The overall capacity check for customers' demands and vehicles' capacity was valid for the manual solution. But by assigning roughly an even number of customers to all depots, it may have led to more individual routes of vehicles than the computational solution and thus to higher set up costs.

As far as available, the routing of the computational and the manual solution also indicate the superiority of the computational to the manual solution: The routes are not comparable since the order of approached nodes is different. But it is already noticeable that the computational solution manages the same number of nodes with fewer vehicles due to a better workload of the vehicles' capacities for the customers' demands.

It should also be noted that the computational solution was already found after about 43 minutes of computing time while after one hour of manual solving, only the assignment was completed.

This validates and verifies the developed code and thus it is further developed.

4.2.3 Outlook

Based on the second implementation and the proposed solving process an outlook is given in this chapter on possibilities to improve and complete the second version of the implemented code.

A lot of requirements described in the outlook of the previous implementation (see chapter 4.1.3) have been successfully implemented. The code is in a stage where it is working well but can be refined and functional features remain not implemented, i.e., the stochastic inputs and the variability of inventory levels as described in chapter 4.1.1.

The malfunction of the cost function should be fixed to work as intended. It would be beneficial to create the possibility of assigning fixed and variable cost to every vehicle so the individual mileage and other individual costs can be measured correctly, depending on their size and other features. This functionality would respond to the intended implementation of a heterogeneous fleet. The cost function could be extended to include costs for additionally purchased products at the warehouse, depending on the purchased amount, when including the warehouse in a route.

The inclusion of the warehouse in the routing could also be improved. In the second implementation, a 2-opt algorithm with a local search procedure is used for improving the routing and also the inclusion of a warehouse into a single route. The disadvantage of this algorithm is that it can get stuck in a local optimum and has no functionality to overcome it. This algorithm could be replaced with a biased randomization with a perturbation of the computed solutions to compute the routes and to find the node that is approached first after the warehouse. This algorithm is more likely to find global optimums and thus increase the quality of the computed solutions.

When using the capacitated round-robin process to assign customers to depots, only the overall capacity of products is considered but not individual products when deciding which depot gets picked next for the assignment. This can lead to an assignment of customers without sufficient stock of individual products at the depot to serve them. A solution could be to consider every product individually during the assignment step.

To make the algorithm more realistic, as already described in chapter 4.1.3, time windows or the maximum of distances could be introduced as factors leading to route failure when violating previously set maximum constraints. Connected to this feature it would be useful to introduce a tool to compute exact distances between map locations instead of using the

Euclidean distance. This would refine the computing of costs and would be a mandatory feature when calculating time windows and/or maximum route distances.

The proposed changes only serve as ideas on how to further improve the implementation and to make the solving process more realistic. It is most likely that not all proposed changes can be implemented due to the already high complexity of the IRP and the solving process. They serve as impulses for further improvement.

4.3 Final Implementation

The final implementation was done based on the previously described ones and written by Lena Pfeilsticker. The basic structure of the processes and steps is not changed but the single steps are refined and include several adaptations as planned in the outlook in the previous chapter. In the following sections, the solving process is described (4.3.1) and two final case studies conducted (4.3.2), a deterministic and a stochastic one, based on the final implementation. To avoid redundancies, not the whole structure of the solving process is described again but the changes conducted compared to the previous implementations described in the chapters 4.1 and 4.2.

4.3.1 Solving Process

After refining the routing procedures in the previous implementation, the focus in the final implementation is on adding stochastic input data and possible adaptations of the depots' individual inventory to minimize overall costs for the inventory routing system. Also, some errors are fixed and some functionalities of the code are refined. The basic structure of the solving process remains while the conducted changes are explained as follows.

The biggest change of the codes' structure is the implementation of being able to generate stochastic input values as customers' demands and to set the depots' inventory levels based on the resulting overall stochastic demands of their assigned customers. The implementation of this functionality increases the algorithm's complexity, thus this part was implemented during the last steps of writing the code. The two parts are implemented simultaneously. This is due to the influence they have on one another. When the customers' demands are deterministic, the overall customer demand that needs to be supplied from one depot can be estimated and determined previously based on the vehicles' capacities assigned to a depot. Thus the inventory levels can be estimated almost exactly and can be set at one level. When the demands are stochastic, the significance of the inventory level increases substantially. Customers' demands cannot be estimated reliably due to their variation. When demands are stochastic and the customers are supplied, the demand may exceed or undershoot the set inventory level. This can either lead to route failures and consequentially to high routing costs for additional routes for getting additional products or on the other hand to high stock costs when the inventory level exceeds the demand. So the inventory levels need to be determined while maintaining minimal costs consisting of routing and stock costs. This can be accomplished using simulation with a distribution function for the customer demand reflecting its fluctuation.

In the present work the depots' inventory levels are modeled using a lognormal distribution with $E[D_{in}] = d_{in}$, where d_{in} represents the fixed demands for each depot i and product n and

D_{in} the probabilistic demand respectively. The distribution type is interchangeable as long as it has a mean. It was chosen based on its use in the related research area of routing problems, especially in the field of distributing products in a gas or liquid state. Three different levels of variance are considered: $Var[D_{in}] = k\, d_{in}$ with $k = 0.25$ defining a 'low' variance scenario, $k = 0.5$ defining an 'intermediate' variance scenario and $k = 0.75$ defining a 'high' variance scenario. For every run, the random customer demands are generated once.

For the customer assignment, the first step of the algorithm, the customer demand is deterministic. When the algorithm has performed the customer assignment and the complete map of customers, depots and the warehouse is depicted, a calculation of the demand each depot has to serve of every product can be performed. This reflects the knowledge a planner has of the product amount to be distributed before arriving at the customers' locations. Based on this calculation the overall amount of requested products is used as a mean value for the lognormal distribution. For the products, a 'product key' is built reflecting the share one product has in the overall demand of one depot. The lognormal distribution is calculated applying different variances as previously described. Following this different inventory levels are set according to five different policies, representing the percentage of the inventory's filling level p = {0.0, 0.25, 0.5, 0.75, 1.0}. These percentages are multiplied with the maximum capacity which is twice of the expected demand and the inventory levels for each policy are set. The values can easily be modified, the approach serves to provide a first indication on possible results.

During a pre-determined number of simulation runs, the routing and stock costs are calculated for every policy. Different stock cost rates are applied to simulate different scenarios. The inserted values are λ = {0.01, 0.25, 0.5, 0.75, 1.0}. The higher the inventory level, the higher the stock costs but also the lower the routing costs to the warehouse due to little to no additional products that need to be purchased to fulfill 100% of customers' demands. The best policy for each individual depot is applied to set the inventory level. To ascertain the individual products' inventory levels, the product key is applied to the overall amount of product. In the algorithm, the routing process is then performed based on the new inventory levels. During this routing process, a new random customer demand based on the same probability distribution is generated. This enables generating different values from the ones the refill policies were decided from and to simulate how well the program can cope with the fluctuation.

This functionality makes the code more realistic and easily adaptable to different strategies used as input.

Furthermore, the possibility of using exact distances between locations of the given map was implemented by Gabriel Alemany. It can be decided beforehand in the input if the distances are computed using the Euclidean distance or if exact distances are employed. The exact distances need to be entered. The disadvantage is, that the more locations (customers, depots and warehouses) are used, the more distances need to be entered (between all map locations).

Also, the possibility of inserting maximum distances that can be driven for all used vehicles was introduced by Gabriel Alemany. This makes the algorithm more realistic and enables to estimate maximum times for driving one vehicle by considering the used roads and determine a mean value for the driving speed. When none is given, the vehicle can drive an unlimited distance.

Another improvement is the possibility of modifying inputs which allows setting the fixed and variable routing costs for every vehicle type.

4.3.2 Case Studies

The final case study conducts a comparison between the functioning of the deterministic code version before the adaption and the stochastic version afterwards. The results are validated and verified with regards to their computation of solutions and costs. First the deterministic version is described and subsequently the testing of the stochastic version.

4.3.2.1 Deterministic Series of Tests

The changes conducted in the code before the adding of the stochastic part are listed as follows:

- A modification of inputs to set the fixed and variable routing costs.
- The possibility of inserting a maximum distance for all used vehicles.
- The possibility of using exact distances between all map locations.

All further changes described in the solving process apply to the stochastic series of tests in the following chapter. For the testing of the deterministic code, provided by Gabriel Alemany, the input data and its modifications are as subsequently described.

- The number of customers is set at i = 100.
- The number of depots is set at d = 3.
- The number of warehouses is set at k = 1.
- The fixed set up costs for sending one vehicle on one route amount to 110 monetary units for the small, 120 for the medium and 130 for the large vehicle.
- There is no set maximum distance for any employed vehicle.
- The variable distance-based costs are set at 0.5 monetary units per unit of length.
- The numbers of products is set at n = 4.

The complete customer inputs for this series of test are similar to the inputs used in the second case study and are listed in Annex 3. Also, again not all vehicles are able to pick up new products at the warehouse. In the following Table 14 the depot inputs are displayed.

Depot ID	X-Coordinate	Y-Coordinate	Capacity Product 1	Capacity Product 2	Capacity Product 3	Capacity Product 4
101	29	21				
102	154	19		p * expDem		
103	64	62				

***Table 14** Inputs Depots Deterministic Final Code*

For every product n, the capacity in every depot is set according to the chosen policy p and the expected demand ‘expDem’ as determined prior to the test runs. This calculation is explained by companies orientating the size of their depots according to the expected demands that are fulfilled from each depot respectively. This approach allows a direct comparison of the effect of

insufficient product capacity in comparison to additional distances to be covered by vehicles since the routing process is performed by also considering the depots' inventory levels.

The considered policies are set at p = {0.0, 0.25, 0.5, 0.75, 1.0} and the stock cost rate at λ = {0.01, 0.25, 0.5, 0.75, 1.0}. This leads to the inventory capacities as displayed in Table 15.

	Policy 0.0	Policy 0.25	Policy 0.5	Policy 0.75	Policy 1.0
Depot 1	-	3,750	7,500	11,250	15,000
		3,750	7,500	11,250	15,000
		3,750	7,500	11,250	15,000
		3,750	7,500	11,250	15,000
Depot 2	-	3,750	7,500	11,250	15,000
		3,750	7,500	11,250	15,000
		3,750	7,500	11,250	15,000
		3,750	7,500	11,250	15,000
Depot 3	-	6,250	12,500	18,750	25,000
		6,250	12,500	18,750	25,000
		6,250	12,500	18,750	25,000
		6,250	12,500	18,750	25,000

Table 15 *Deterministic Inventory Levels for all Depots and Policies*

The number of the depots' vehicles and associated capacities and number of compartments remains, the values can be looked up in chapter 4.2.2.

The warehouse is described with its ID and the x- and y-coordinate in Table 16.

Warehouse ID	X-Coordinate	Y-Coordinate
104	60	36

Table 16 *Warehouse Inputs Final Deterministic Test*

For the computation of results the run time was set at 45 min for every one of the five tests due to no improvement of previously computed solutions between minute 45 and 60 during the run time and the number of tests to be performed. The inventory levels were calculated prior to the testing based on the expected deterministic customer demand per depot. The number of computed solutions ranged between 90,000 and 98,000.

The overall customer demand accumulated over all products amounts to 170,064 products. The costs for the remaining inventory are computed at the end of each period after all shipments are completed. That leads to the following inventory costs for each policy in Table 17.

		Policy 0.25	Policy 0.5	Policy 0.75	Policy 1.0
Accumulated Inventory	Depot 1	15,000	30,000	45,000	60,000
	Depot 2	15,000	30,000	45,000	60,000
	Depot 3	25,000	50,000	75,000	100,000
Overall Inventory		55,000	110,000	165,000	220,000
Difference to overall customer demand		-115,064	-60,064	-5,064	49,936
Inventory Costs λ	0.01	0	0	0	499.36
	0.25	0	0	0	12,484
	0.5	0	0	0	24,968
	0.75	0	0	0	37,452
	1.0	0	0	0	49,936

Table 17 *Inventory Costs per Policy*

The complete results of this series of tests are summarized in Table 18. The abbreviations stand for R: routing costs, I: inventory costs, T: total costs.

	Policy 0.0	Policy 0.25	Policy 0.5	Policy 0.75	Policy 1.0
Lambda 0.01	R: 2,733.38 I: - T: 2,733.38	R: 2,551.86 I: 0 T: 2,551.86	R: 2,511.82 I: 0 T: 2,511.82	R: 2,503.50 I: 0 T: 2,503.50	R: 2,503.50 I: 499.36 T: 3,002.86
Lambda 0.25	R: 2,733.38 I: - T: 2,733.38	R: 2,551.86 I: 0 T: 2,551.86	R: 2,511.82 I: 0 T: 2,511.82	R: 2,503.50 I: 0 T: 2,503.50	R: 2,503.50 I: 12,484 T: 14,987.5
Lambda 0.5	R: 2,733.38 I: - T: 2,733.38	R: 2,551.86 I: 0 T: 2,551.86	R: 2,511.82 I: 0 T: 2,511.82	R: 2,503.50 I: 0 T: 2,503.50	R: 2,503.50 I: 24,968 T: 27,471.50
Lambda 0.75	R: 2,733.38 I: - T: 2,733.38	R: 2,551.86 I: 0 T: 2,551.86	R: 2,511.82 I: 0 T: 2,511.82	R: 2,503.50 I: 0 T: 2,503.50	R: 2,503.50 I: 37,452 T: 39,955.50
Lambda 1.0	R: 2,733.38 I: - T: 2,733.38	R: 2,551.86 I: 0 T: 2,551.86	R: 2,511.82 I: 0 T: 2,511.82	R: 2,503.50 I: 0 T: 2,503.50	R: 2,503.50 I: 49,936 T: 52,439.50

Table 18 *Overview results for deterministic Series of Tests*

As can be noted the routing costs are similar for every stock cost rate λ, since the inventory costs are computed based on the previously set inventory levels and thus do not influence the routing process. Only in p = 1.0 arise stock costs. Also notable is: the lower the inventory level, the higher the routing costs as should be expected. This validates the developed algorithm.

4.3.2.2 Stochastic Series of Tests

The stochastic computation is based on a series of trials to determine a number of customers with which the program can run and produce a sufficient number of feasible solutions. This

series of trials was conducted previous to the actual testing. The trial runs led to the following inputs.

- The number of customers is set at i = 20.
- The number of depots is set at d = 3.
- The number of warehouses is set at k = 1.
- The fixed set up costs for sending one vehicle on one route amount to 110 monetary units for the small, 120 for the medium and 130 for the large vehicle.
- There is no set maximum distance for any employed vehicle.
- The variable distance-based costs are set at 0.5 monetary units per unit of length.
- The numbers of products is set at n = 4.

The complete customer inputs for this series of test are listed in Annex 5. In the following Table 19 the depot inputs are displayed.

Depot ID	X-Coordinate	Y-Coordinate	Capacity Product 1	Capacity Product 2	Capacity Product 3	Capacity Product 4
101	29	21				
102	154	19		$2 * \sum_{1}^{i} \sum_{1}^{n} (E[D_{in}] = d_{in}) * bestPolicy$		
103	64	62				

Table 19 *Inputs Depots Stochastic Code*

The capacities of the depots are computed based on the accumulated estimated customer demand that occurs in every sub-map. Every sub-map is associated with one depot and a number of customers assigned to it. The number of depots and the single customers vary at every iteration of the program. During the testing with stochastic value the estimated customer demand is computed based on a known probability distribution with $E[D_{in}] = d_{in} > 0$. The approach of doubling the expected customer demand for setting the inventory capacity of one depot serves to assure the fulfillment of 100% of customer demands with stochastic values. Three different variances are tested determining a low, medium and high variance scenario as described in chapter 4.3.1. The estimation of customer demands serves to simulate and determine the refill policy with the lowest overall cost. The multiplication of all factors determines the inventory level for every product of every depot. The considered policies are again set at p = {0.0, 0.25, 0.5, 0.75, 1.0} and the stock cost rate at lambda λ = {0.01, 0.25, 0.5, 0.75, 1.0}. The number of the depots' vehicles and associated capacities and number of compartments remains, the values can be looked up in chapter 4.2.2.

The warehouse is described with its ID and the x- and y-coordinate in Table 20.

Warehouse ID	X-Coordinate	Y-Coordinate
24	60	36

Table 20 *Warehouse Inputs Final Stochastic Test*

For the computation of results the run time was set at 45 min for every one of the 15 tests due to no improvement of previously computed solutions between minute 45 and 60 during the run time and the number of tests to be performed. The number of computed solutions ranged between 1,000 and 5,000.

For the stochastic testing the best solution policies for every depot and the best solution routing and stock costs for every sub-map are calculated and listed according to the different stock cost rates and variances. The results are displayed in the following Table 21. P stands for the policies set as best ones for depot 1, 2 and 3 in the simulated scenario.

	Variance 0.25	Variance 0.5	Variance 0.75
Lambda 0.01	P: 0.5; 0.5; 0.5 R: 1,401.35 I: 1,186.15 T: 2,587.50	P: 0.5; 0.5; 0.5 R: 1,423.37 I: 1186,15 T: 2,609.52	P: 0.5; 0.5; 0.5 R: 1,391.91 I: 1,186.15 T: 2,578.06
Lambda 0.25	P: 0.5; 0.5; 0.5 R: 1,401.35 I: 29,653.75 T: 31,055.10	P: 0.5; 0.5; 0.0 R: 2,011.70 I: 14,044.25 T:16,055.95	P: 0.5; 0.5; 0.0 R: 2,540.28 I: 7,869.00 T: 10,409.28
Lambda 0.5	P: 0.5; 0.5; 0.0 R: 1,605.51 I: 15,738.0 T: 17,343.51	P: 0.0; 0.5; 0.0 R: 1,850.70 I: 15,738.00 T: 17,588.70	P: 0.0; 0.5; 0.0 R: 1,858.07 I: 15,738.00 T: 17,596.07
Lambda 0.75	P: 0.0; 0.5; 0.0 R: 1,841.69 I: 23,607.00 T:25,448.69	P: 0.0; 0.5; 0.0 R: 1,850.70 I: 23,604.00 T: 25,457.70	P: 0.0; 0.0; 0.0 R: 1,765.09 I: 0.0 T: 1,765.09
Lambda 1.0	P: 0.0; 0.5; 0.0 R: 1,841.69 I: 31,476.00 T: 33,317.69	P: 0.0; 0.0; 0.0 R: 1,820.07 I: 0.0 T: 1,820.07	P: 0.0; 0.5; 0.0 R: 1,841.69 I: 31,476.00 T:33,317.69

Table 21 *Overview results for stochastic Series of Tests*

The results are summarized in Figure 20 and Figure 21. Figure 20 displays the proportions of routing and inventory costs for all variances and stock cost rates.

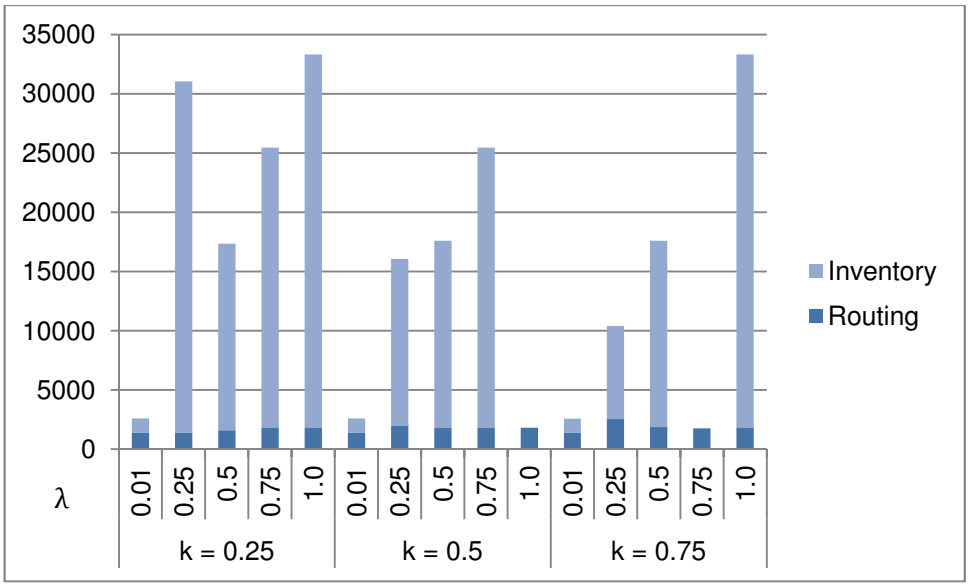

Figure 20 *Proportions of Stochastic Routing and Inventory Costs*

The results reflect the costs adequately, the higher the stock cost rate λ, the higher the overall inventory cost. Furthermore, in the case of p = 0.0 in all depots, no inventory costs are incurred or if one or two depots out of three have an inventory set at p = 0.0, so the overall inventory costs are lower respectively. It needs to be considered though that the individual proportions of inventory to routing costs are not useful in the most cases in Figure 20. The inventory ratio exceeds the routing ratio by far.

The results of the inventory costs with different variance levels become more balanced and predictable, the higher the variance is set. While at k = 0.25 the inventory costs are fluctuating strongly, at k = 0.75 the inventory costs rise consistently with the increase of λ. This contradicts the assumption of creating a highly unbalanced scenario by increasing k. On the other hand the sample size from which the results were computed may not be sufficiently large to reflect the outcome of different variance scenarios.

Further analysis on the set inventory levels led to the results displayed in Figure 21.

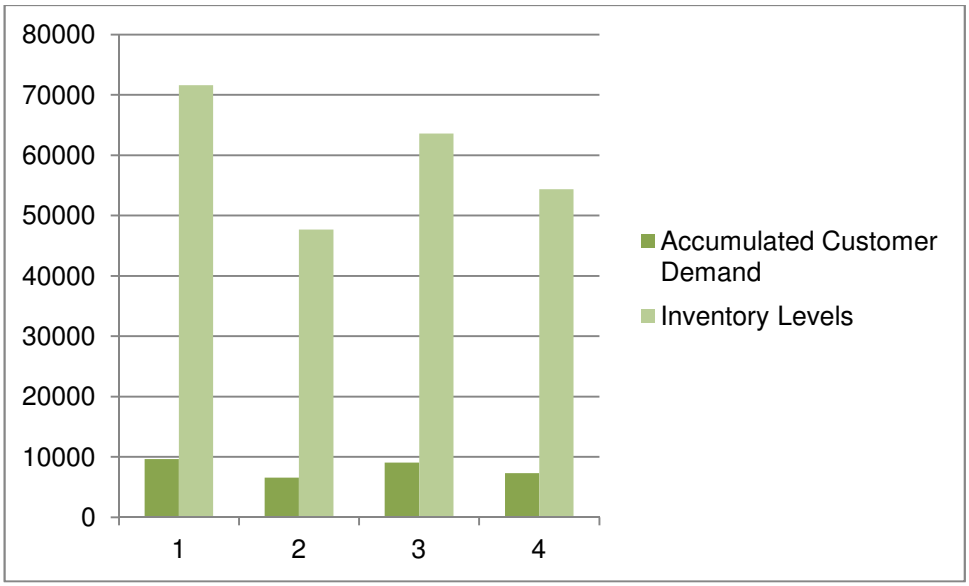

Figure 21 *Comparison of Customer Demand and Inventory Levels*

It shows a high discrepancy between the accumulated mean customer demand and the related inventory levels per product. The algorithm should set the inventory level according to the estimated customer demands with a given mean and a variance. This process is not performed correctly and leads to high stock costs.

In Figure 22 the proportions of routing costs of different stock cost rates and variances are displayed.

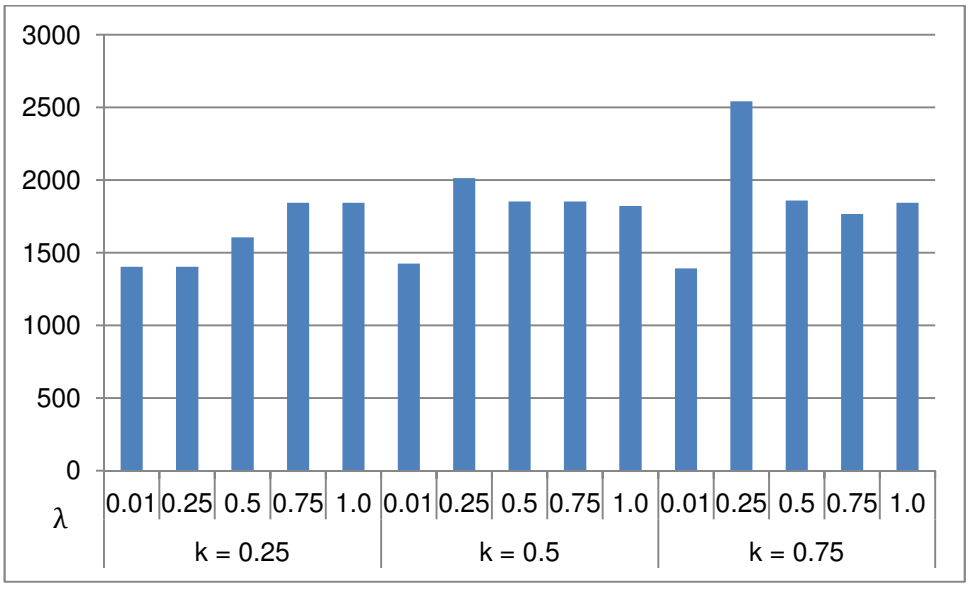

Figure 22 *Routing Costs Stochastic Solution*

The routing costs and their proportions are displayed separately for further analysis. As should be expected, the routing costs rise with a higher stock cost rate due to more refill stops at the warehouses leading to longer distances driven by the used vehicles. It can be noted that in the

higher variance scenarios, the routing costs are not balanced and predictable as in the low variance scenario. This is what can be expected with the given initial situation of variances.

4.3.3 Risk Analysis

Risk analysis is a way of assessing the risk linked to acting a pre-defined way. To achieve the risk that is connected to a certain course of action, simulation of different scenarios and a comparison of the outcome is a possibility with reasonable results.

The risk analysis is performed for the deterministic solution as well as for the stochastic solution.

The risk analysis for the deterministic code is based on the results displayed in Table 18. Due to the size of the inventory costs that only arise when applying p = 1.0, only the routing costs are considered and analyzed. The ratios of routing costs associated with every policy are displayed in Figure 23.

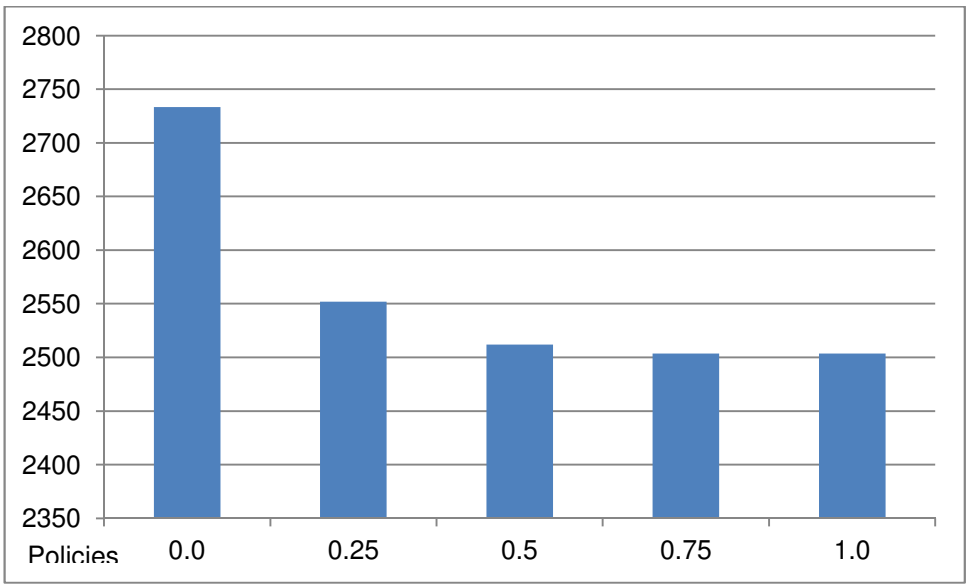

Figure 23 *Routing Costs of the deterministic Solution*

It should be noted that even though p = 0.75 and p = 1.0 have the lowest routing costs, p = 1.0 has high inventory costs and exceeds all other policies' costs. Thus in this scenario the best option would be choosing the p =0.75 without stock costs and the lowest routing costs.

The risk analysis for the stochastic code is based on the results displayed in Table 21. The mean of the accumulated overall costs for every variance scenario build the base for the boxplot displayed in Figure 24. It contains a logarithmic trendline, following the underlying probability distribution. The boxplots are displayed for every variance scenario with the overall computed costs. The figure is not an instrument in a decision-making process but can be used to quantify the costs associated with a known variance.

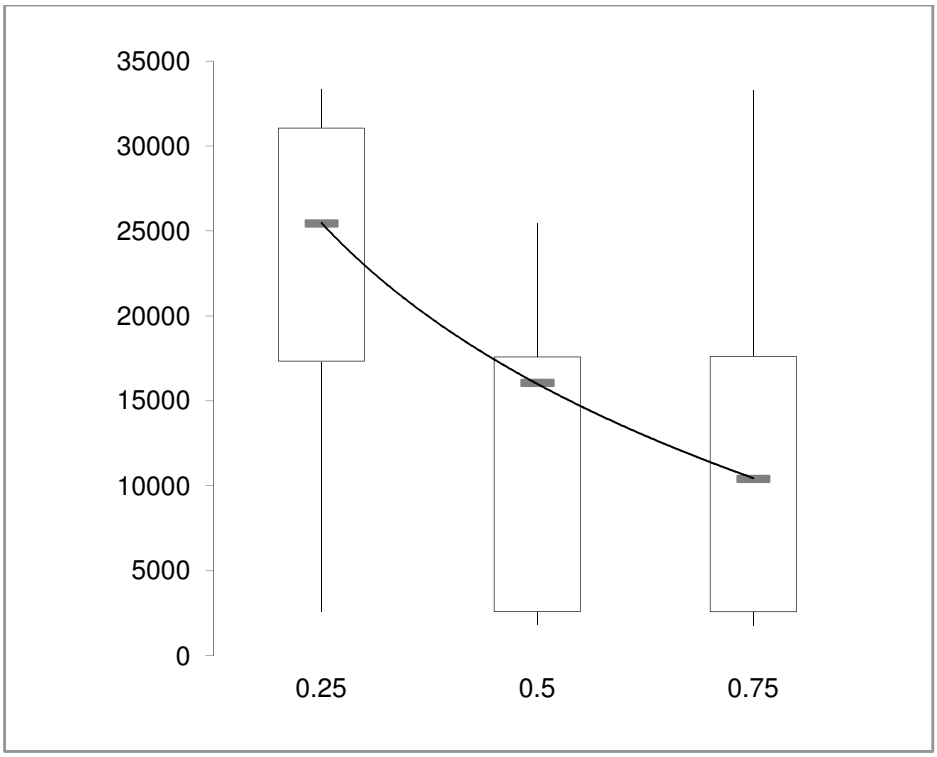

Figure 24 *Boxplot of the Stochastic Results*

The grey bars represent the median of every variance scenario. As can be noted, the costs in a low variance scenario are about double compared to the other scenarios. When comparing the scenarios with the variances 0.5 and 0.75, the overall costs seem to be similar but the median of the 0.75 variance scenario is lower and thus is the most beneficial scenario.

This result contradicts the assumption that the higher uncertainties, the higher the cost due to less assessability of customer demands.

The algorithm determined a varying mix of p = 0.5 and p = 0.0 as best solutions, depending on the stock cost rate. To decide for a refill policy, Figure 20 clearly indicates the lowest costs when no stock exists. But the risk connected to no inventory while having stochastic customer demands that need to be fulfilled 100% every period is substantial. The fluctuations can lead to insufficient transport capacities, route failures and thus high costs. The policy p = 0.0 is only realistic to be implemented when the customer demands can be quantified very accurately to prevent high out-of-stock-costs or high inventory costs.

4.3.4 Outlook

The program is not further developed in the present work but possibilities on advancing are given in this chapter to determine components that can be improved to produce better results.

As described in the analysis of the stochastic code, the inventory levels do not reflect the customer demand and lead to excessive stock costs due to high and unnecessary inventory levels. The calculation of customer demands in the program is performed correctly as has already been reviewed so the setting of the new inventory levels in the single depots seems to be incorrect. This can be corrected but it was not possible in the present work as it would have

exceeded the given time frame. Then, additional simulation runs can serve to determine the advantage of using a safety stock. This approach would be interesting in combination with a multi-period IRP to determine the most advantageous point (in time or inventory amount) of replenishing the inventory. Especially when doing research in the field of delivering gas, fuel and other liquids of this kind, one approach to accomplish more precise forecasts for replenishment strategies is installing measuring devices in the customers' tanks. This way, when a pre-determined filling level is reached, the device signals this to the supplier and the customer's inventory can be replenished. Another approach could be to install a device that exactly documents the filling level of a tank at all times. Thus, when planning routes, it can be considered to supply customers without immediate need for the product but with a sufficiently empty tank, that are located along the route. This way routing costs could be saved.

In the final implementation the stochastic customer values are computed once and are then set for the whole simulation process. This can be changed to vary the customer demands to be different at every iteration of the program.

Another feature that would make the program more realistic is introducing a fee for products purchased at the warehouse when the inventories at the depots are not sufficiently stocked. This would change cost calculations because including a trip to the warehouse so far only included the additional distance costs. In a realistic scenario an additional product purchase concludes additional costs.

As mentioned in previous outlooks, the inclusion of time windows is not performed in the present work but would make the approach even more realistic. The complexity of including time windows in the algorithm though exceeds the given time frame. The specifying of maximum distances for vehicles with the inputs however can serve as an instrument to model approximate time windows. This can be conducted when the mean travelling speed between different locations is known.

The possibility of inserting exact distances is provided in the final code while maintaining the possibility of applying Euclidean distances with provided locations in a coordinate system. A disadvantage in the final code though is the necessity of inserting the exact distances manually between all locations on the employed map. When dealing with realistic scenarios, a set goal of the present work, it is inevitable to implement a tool that can provide exact distances between locations automatically. Otherwise, when inserting the exact distances, the necessary number of distances between locations would be n!. In a system of 1,000 customers and more, this is not applicable in a reasonable time frame.

As described in the previous outlook, an improvement for a higher level of simulation, the selection of the warehouse and its placement in a route can be biased randomized for achieving better routing results and thus lower overall costs.

5 Technical Description of the Final Code

This chapter aims at giving a detailed overview of the developed code's functionalities and its scope. The final code consists of seven packages (main, algorithm, data, map, models, plot and RCWS) with 30 classes. The classes were written and modified by Enoc Martinez, Gabriel Alemany and Lena Pfeilsticker.

When parts of the code are shortened, '…' indicates an exclusion of code.

5.1 Input

Two input files are necessary to run and test the code. The first file (test2run) includes the memory location of the test to run and several parameters. An example of the file is given in Figure 25.

```
# name|beta|savingParam|roundRobinFraction|seed|time
test 0.3 0.7 1.0 20213 60
```

Figure 25 *Input files Part I*

The first line describes the values in the following. The variable 'name' indicates the file which contains the input for the tests. The variables 'beta' and 'savingParam' are passed to the class 'RNG' of the package main. They are used to determine the geometrical distribution for the computation of the solutions of the mapping and routing procedures in the method 'getNextIntGD(int n, double beta)' as displayed in Figure 26.

```
    public static int getNextIntGD(int n, double beta) {
       int pos;
       double d = _rng.nextDouble();
         pos = (int)(Math.log(d) / Math.log(1 - beta));
         pos = pos % n;
         return pos;
    }
```

Figure 26 *Public Method 'getNextIntGD' of class 'RNG'*

In this work a lognormal distribution is employed.' *_rng*' is an instance of the class 'Random', which is used to generate a stream of pseudorandom numbers.

The result of calling the method 'getNextIntGD(int n, double beta)' during each mapping or routing procedure is the determination of the distribution's width which translates to the amount of computable different solutions. More precisely, the bigger 'beta' or 'savingParam' are, the lower the variance of solutions, so the range of computed solutions is smaller. This leads to a usage of the same solutions with a higher frequency and this to a lower number of possible

combinations of mapping and routing options. So the search space is not explored comprehensively but leads to feasible solutions in a shorter time because the program does not need as much time to explore possible solutions. But in case no or few solutions are found, ‘beta’ and ‘savingParam’ should be lowered in order to widen the search space. Simultaneously the program’s run time should be extended due to a higher amount of possible solutions.

The variable ‘roundRobinFraction’ in Figure 25 is used during the assignment step and indicates the proportion of a vehicle’s load that has to be assigned in phase 1 of the round-robin process. In other words, it describes the process of assigning the equitative theoretical load of every sub-map multiplied with the variable ‘roundRobinFraction’ during phase I of the assignment of customers to depots. To finish the assignment step in phase 2, the remaining customers are assigned based on the highest overall capacity left in the comparison of all available depots.

The variable ‘seed’ in Figure 25 serves to simulate the stochastic behavior of a distribution in a program. When simulating stochastic distributions in a program to compute random numbers however, the computed values are not stochastic due to computer systems being deterministic machines. For this reason a (pseudo-)random number RNG is employed to model stochastic behavior. The given seed is a number used to initialize the RNG. It starts with an initial seed, an integer number defined previously to the simulation runs. From this seed, a function creates a stream of ‘random’ values. Using the same seed value always computes the exact same stream of numbers. Due to this it is a pseudo-random number generator. A sample of a distribution uses one or more numbers from the stream to compute its value. The value of the initial seed thus decides the value of all samples drawn in the simulation. At the end of the simulation, the used initial seed of that simulation is printed for reference purposes.

During the conducted case studies the variables ‘beta’, ‘savingParam’, ‘roundRobinFraction’ and ‘seed’ have not been changed during individual series of tests. The experiments on the variables’ impact on calculated results was not closely investigated since the focus was on other factors.

The last variable‘time’ given in Figure 25 describes the time in seconds and determines the run time the program can use to compute solutions.

The second input file is constructed as displayed exemplary in Figure 27.

```
# customers|depots|WH|diffProducts|diffVehicles|distType|Variance k|stock
cost lambda
6 3 1 4 3 0 0.25 0.1
# vehicle id|set-up costs|var cost rate|WH able|max dist|compartm cap 1 to n
1 110. 0.5 0 0. 2000 2000
2 120. 0.5 1 0. 5500 5500
3 130. 0.5 1 0. 6000 10500 13500
# customer id|x|y|demands product 1 to n
01 61 09 892 329 0 337
02 71 09 656 201 588 542
03 80 09 589 681 704 922
04 90 08 0 54 912 291
05 162 06 761 210 732 318
06 73 81 335 744 289 840
# depot id|x|y|inventory product 1 to n|no of diff veh. types|no of type|type
07 29 21 15000 15000 15000 15000 2 7 1 7 2
08 154 19 15000 15000 15000 15000 2 7 1 7 2
09 64 62 25000 25000 25000 25000 2 7 1 5 3
# warehouse id|x|y
10 60 36
```

Figure 27 *Input files Part II*

This is the input file named as indicated with the first variable in the first input file. The lines starting with '#' are skipped in the code and only serve as comment lines for the following values. In the first line the number of customers, the number of depots, of warehouses, number of different products, the distance type, the number of different vehicles, the variance for the distribution function and the stock cost rate λ are defined.

Every vehicle type used is described in an extra line. Each one is given its ID, the set-up costs, the variable cost rate, if it can drive to the warehouse or not, the maximum distance it can cover in one period and the number of compartments with their individual capacities. If a vehicle is able to drive to the warehouse is indicated with 1 = yes or 0 = no. The maximum distance of a vehicle is a float value. If it is without value (0.) no maximum distance level is given and the vehicle can drive unlimited ways per period. The number of compartments is indicated with the number of different compartment capacities e.g., (2000, 2000) indicates two compartments with a capacity each of 2,000 units.

Customers are described in a similar pattern. For every customer their ID, their location in a coordinate system (x|y) and the size of their demand for every product are printed in one line.

For every depot exists a line in the input file including their ID, their location (x|y), the inventory level for every product, the number of different vehicle types, the number of the first type, the assigned vehicle type ID and the number of the last type and the assigned vehicle type ID, i.e., in line 07 the numbers "2 7 1 7 2" describe the depot having two different types of vehicles, 7 of type 1 and 7 of type 2.

In the last lines the warehouse IDs and their locations (x|y) are given.

5.2 Computer Operations

As previously described the program consists of seven packages with 30 classes. In the following the different packages' functionalities are described.

Package Main: This package contains the class 'RNG' and the class 'TestsRunner' with the main method.

The class 'RNG' manages the random behavior of the program. More precisely the class contains several methods to compute random numbers based on geometrical or uniform distributions. For this purpose a RNG is employed, in this work an instance of the class 'Random'. It is used to generate a stream of pseudorandom numbers.

The class 'TestsRunner' plays a central role in the program. It contains the main method and amongst others uses the instances of the classes 'Algorithm' and 'Solution' that call all methods used to compute the solution. It scans the inputs and provides the data for the output.

Package Algorithm: This package contains the classes 'Algorithm' and 'SubAlgorithm'. The instance of the class 'Algorithm' is instantiated and used by the main method and contains the method solve() which contains the whole solving process. It plays a central role in the running of the code and thus is described in detail. It was written by Enoc Martinez and extended by Gabriel Alemany and Lena Pfeilsticker.

First all savings for the depots are computed, based on the distances calculated in the main method of the class 'TestsRunner'. This process is displayed in Figure 28.

```
// 1. Compute savings for depots (PRELIMINARY)
      Savings savings = MapTools.calcDepotSavings(input);
```

Figure 28 *Computing Savings for Depots*

Afterwards a MS process is started with a given run time to compute solutions. This contains generating a map assignment of customers to depots based on the generated savings. Then the method 'calculateRefillPolicies()' gets the expected demand for every depot and calculates the best inventory policy with lowest overall costs out of a given number of different policies for every depot assuming stochastic customer demands. The calculation is based on the minimization of stock and routing costs. These steps are performed in the instance of the class 'SubAlgorithm' (written by Lena Pfeilsticker). In Figure 29 the process is displayed with the core methods.

```
// 2. Multi Start Process
…
    // 2.1 Generate map assignment

            CompleteMap map = genRandAssignmentMap(nodes, depots, savings);

        //2.1.1 Get expected demand for every depot
        …
        //2.1.2 Calculate policy with the lowest overall costs for each depot
            SubAlgorithm subAlg = new SubAlgorithm(input, map);
            subAlg.calculateRefillPolicies();
            …
```

Figure 29 *Multi-Start Process Part I*

Then each map is solved applying a biased randomized CWS heuristic. Subsequently the total costs of routing and stock are computed. The amount is compared to the variable 'bestSol' which represents a type of cache and saves the best computed solutions of the program. The variable 'bestSol' is updated when a new best solution is found and printed as overall minimal cost found in the given time frame.

The variable 'bestSolDepots' contains a list of all depots and saves their properties' values whenever a new best solution is found. It enables to print out the values of every depot of the best found solution, e.g. the inventory levels. This process is displayed in Figure 30.

```
    // 2.2 Solve each map
    …
            SubSol partialSol = biasedRandRichCws(sm, input);
            …
        //Calculate total costs of routing and stock
        newSol.CalcTotalCosts(map);

    // 2.3 Compare with bestSol
    …
                if (bestSol == null || newSol.getTotalCosts() <
                bestSol.getTotalCosts()) {
                    bestSol = newSol;
                    bestSolDepots = in.getDepots();
                    …
                }
```

Figure 30 *Multi-Start Process Part II*

The method 'biasedRandRichCws(Submap sm, Inptut input)' creates a sub-solution for every sub-map consisting of routes to supply all customers in one sub-map. First all customers are converted into several sub-customers, equal to the number of demanded products. Then the routing procedure is performed with the sub-customers. Afterwards the sub-customers are converted back into the original customers again. This serves to enable the visit of one customer by more than one vehicle und simplifies the implementation in the code. This process is displayed in Figure 31.

```
    private SubSol biasedRandRichCws(SubMap sm, Input input) {
        List<Vehicle> vehicles = sm.getDepot().getVehicles();
        Random rng = test.getRng().getrng();
        …
    // Create a list with one sub-customer for each customer's demand of
    products

        List<SubCustomer> subcustomers = customers2subCustomers(sm);
        List<Route> goodRoutes = new LinkedList<Route>();
        …
    // Convert the SubCustomers route to a Customers one
                r = subcustomers2customers(r);
                goodRoutes.add(r);

    return constructSubSolution(goodRoutes, sm, input.getDiffProduct());
    }
```

Figure 31 *Routing for sub-customers*

As previously described, the round-robin process of assigning customers to depots is split into two phases. The variable 'roundRobinFraction' is used to determine the amount of every depot capacity that is assigned in the first phase of the round-robin process. When the amount of capacity to be assigned is reached, phase two starts and chooses sub-maps with higher capacity levels over others. This process is displayed in Figure 32.

```
double factor = test.getfRoundRobin() * (double)input.getTotalDemand() /
(double)input.getTotalVehCapacity();
…
// Phase I: Round-Robin until all submaps reach their
// equitative theoretical load multiplied by parameter roundRobinFraction
…
    Collections.sort(subMapList0); // By capacity in depot vehicles
                …
                boolean isLastNodeOfDepot = mapper.assignBestNode(sm, nodes);
                if (isLastNodeOfDepot) {
                    subMapList0.remove(sm);
                    subMapList.remove(sm);
                } else
                    if (sm.sumOfAllStatus() > factor *
                        sm.getSumOfAllVehCapacitys())
                        subMapList0.remove(sm);
                    …
// Phase II: priorization of submaps with the highest transport capacity left
…
          Collections.sort(subMapList); // By free capacity in depot vehicles
          boolean isLastNodeOfDepot = mapper.assignBestNode(sm, nodes);
          …
```

Figure 32 *Assignment with a (capacitated) Round-robin process*

The other class of the package algorithm is 'SubAlgorithm'. It is used and instantiated by the instance of 'Algorithm' and serves to generate stochastic demands for setting the best inventory policy for each depot. The generation of the stochastic demands is performed independently of

the generation of the individual stochastic customer's demands. When planning the vehicle load before the routing process, a planner cannot know the exact demand but has to anticipate the total customer demand based on a distribution. For this a lognormal distribution is used as described in chapter 4.3.1 and the stochastic demands are calculated for every sub-map. This process is depicted with the code as displayed in Figure 33. It is based on the cumulated expected demand of all products and customers as mean value E (variable 'mean' in Figure 33), the calculation of the variance σ^2 (variable 'sigma' in Figure 33) and the expected value μ (variable 'mu' in Figure 33). They are defined as follows:

$$\sigma^2 = \ln\left(\frac{Var}{E^2} + 1\right)$$

$$\mu = \ln(E) - \frac{\sigma^2}{2}$$

The variance and the expected value are those of the normal distribution. It can be applied to a lognormal distribution when the mean value and the required variance of it are known. Thus the variable 'var' in Figure 33 designates the variance of the lognormal distribution while 'sigma' refers to the variance of the normal distribution.

The value of 'var' is calculated using the mean and the variable 'k' that is passed to the program with the inputs. With the instances of the classes 'Distribution' and 'RandomVariateGen' a random number of a lognormal distribution is created with the given σ and μ. The instance 'stream' is based on a stream of random numbers. This enables the use of the instance 'rngLogN', of the class 'RandomVariateGen', to generate a random expected demand.

```
…
    // 1.1. Set the random distribution for this depot.
    double mean = aDepot.getExpDemand();
    double var = in.getK() * mean;
    double factor = Math.log(1 + var / (mean*mean));
    double mu = Math.log(mean) - 0.5 * factor;
    double sigma = Math.sqrt(factor);
    Distribution dist = new LognormalDist( mu, sigma );
    RandomVariateGen rngLogN = new LognormalGen( stream, (LognormalDist)
   dist );
```

Figure 33 *Generate a Stochastic Total Customer Demand*

During the next step a number of simulation runs is performed based either on a given run time or a number of iterations.

In the following, to calculate the best policy, the maximum capacity of every depot is set at double the cumulated overall expected demand of a sub-map. The code also allows setting a current level of all products for every depot but this feature is not used for the tests in this work.

The arrays 'policiesByRefill' and 'policiesByCosts' are set with new arrays of policies that are filled in with the same values at the beginning. The array 'policiesByRefill' is used to calculate the different policies to be tested and assigned to the various depots. The policies are calculated in a for-loop. The values can be changed as required and at this stage only are exemplary calculated. This process is displayed in Figure 34.

```
// 1.2. Perform simulation runs.
    long start = ElapsedTime.systemTime();
    double elapsed = 0.0;
    int nSimIter = 1000000; //can be used with time instead of number of
                            iterations

     maxCap = 2 * (aDepot.getExpDemand()); //sets max cap of a depot to
                                            double the expected demand
     currentLevel = maxCap; //interchangeable to preset level
     policiesByRefill = new Policy[numberOfPolicies];
     policiesByCosts = new Policy[numberOfPolicies];
        for( int h = 0; h < numberOfPolicies; h++ )
        {
            policiesByRefill[h] = new Policy(maxCap, h * 0.25f,
                                    currentLevel * h * 0.25f);
            policiesByCosts[h] = policiesByRefill[h];
        }
```

Figure 34 *Class 'SubAlgorithm': Perform Simulation Runs*

A random demand is calculated as described previously and is used to calculate the total surplus and stock costs associated with every inventory refill policy. In Figure 35, the commented out calculation of surplus is used when an inventory has already a current level that is filled up to the associated inventory level. Since this is not the case, the surplus is calculated simply based on the amount of product in the depot's inventory minus the generated random demand. So the surplus is calculated based on the amount of products left in the inventory at the end of the period. The stock costs are computed based on lambda, determined in the input files, or based on the costs of a roundtrip to the warehouse to replenish the depot inventory. This is a basic assumption but serves the purpose of evaluating the difference in occurring costs. The costs can be easily modified and adapted to other scenarios.

In the code in 1.3 the computed stock costs and the surplus is divided by the number of reruns of the for-loop and the mean values are saved for every policy since they were accumulated. These calculations are displayed in Figure 35.

```
        for( int j = 0; j < nSimIter; j++ )
        {
            // Generate random demand.
            float randomDemand = (float) rngLogN.nextDouble();
            // Compute accumulated surplus and stock costs.
            for( Policy p : getPoliciesByRefill() )
            {
// float surplus = p.getUnitsToServe() + getCurrentLevel() - randomDemand;
// This calculation has to be used when the current inventory levels of
// depots are taken into consideration.

            float surplus = p.getRefillUpToUnits() - randomDemand;
            p.setExpSurplus(p.getExpSurplus() + surplus); // accum. surplus

            double stockCosts;
            if( surplus >= 0 )
                stockCosts = in.getLambda() * surplus;
            else
                stockCosts = aDepot.getRoundtripToWarehouseCosts();

                p.setExpStockCosts(p.getExpStockCosts() + stockCosts);
                // accum. stock costs
            }
        }
        elapsed = ElapsedTime.calcElapsed(start, ElapsedTime.systemTime());

   // 1.3. Compute expected surplus and stock costs.
        for( Policy p : getPoliciesByRefill() )
        {
           p.setExpSurplus(p.getExpSurplus() / nSimIter);//calc. mean value
           p.setExpStockCosts(p.getExpStockCosts() / nSimIter); //divide by
           number of iterations, needs tob e changed when elapsedTime is
           employed
        }
```

Figure 35 *Calculate expected Surplus and associated Stock Costs for every Depot*

Then the computed policies are sorted by costs for every depot and individually set as the best policy. The stochastic expected demand is divided into four products by applying the previously computed product key. Then the inventory levels as set by the values in the input files are replaced with the inventory levels for every product as associated with the best computed policy. These processes are displayed in Figure 36.

```
// 1.4. Sort policies by Costs for every Depot (policiesByRefill will not
    be modified).
    Arrays.sort(getPoliciesByCosts()); //ascending order, best solution is
                                        on position 0

    //1.5 Set best policy for every Depot
    aDepot.setBestSolPolicy(policiesByCosts[0]);

    //1.6 Apply bestPolicy to every Depot and then apply productKey
    aDepot.calculateProductKey(aSubMap);

    //1.7 Overwrite inventory level for every product of every Depot in
    corresponding sub-map
    int[] productKey = aDepot.getProductKey();
    int productKeyTotal = 0;
    int[] productsCapacity = new int[productKey.length];
        for( int o = 0; o < productKey.length; o++){
            productKeyTotal += productKey[o];
         }
        //1.7.1 Set new level for every product by applying productKey to
        maxCap*best policy percentage
            for( int r = 0; r < productKey.length; r++){
                productsCapacity[r] = ((int)Math.ceil(maxCap) *
                (int)Math.ceil(aDepot.getBestSolPolicy().getRefillUpToPercent())
                    * productKey[r]) / productKeyTotal;
            }
        aDepot.setProductsCapacity(productsCapacity);
    }
}
```

Figure 36 *Setting a new Inventory Level for every Product of every Depot*

Package Data: This package contains the classes 'Input', 'Solution' and 'SubSol'. It serves to process the inputs and outputs of the program.

The class 'Input' is instantiated with the input as scanned in 'TestsRunner' of the main package as a parameter. The instance also contains the method 'createStochasticInputs()' to generate stochastic customer demands. This method was written by Lena Pfeilsticker. It is displayed in Figure 37. The process is similar to the one in the class 'SubAlgorithm' of package 'algorithm'. This serves to employ the similar probability distribution for both the calculation of customer demands and the expected demands of the depots in order to be able to compare them. The method replaces the inputs from the input files with new values based on the lognormal distribution with the given variance k. This step is only performed once and customer demands are set for all following calculations.

```
public void createStochasticInputs(){

   Iterator<Customer> iter = getNodes().iterator();
   for( int i = 1; i < nodes.size(); i++ )
      {  Customer aCustomer = iter.next();
      int[] productDemands = aCustomer.getProductDemands();

      for (int prod = 0; prod < diffProduct; prod++){
         double mean = productDemands [prod];
         double var = getK() * mean;
         double factor = Math.log(1 + var / (mean*mean));
         double mu = Math.log(mean) - 0.5 * factor;
         double sigma = Math.sqrt(factor);
         Distribution dist = new LognormalDist( mu, sigma );
         RandomVariateGen rngLogN = new LognormalGen( stream,
         (LognormalDist) dist );

           // Generate random demand.
           int randomDemand = (int) rngLogN.nextDouble();
           productDemands [prod] = randomDemand;
      }
      aCustomer.setProductDemands(productDemands);
   }
```

Figure 37 *Generating Stochastic Customer Demands*

The class ‘Solution’ contains the routing costs and the stock costs as properties and the method ‘calcTotalCosts(CompleteMap map, Input input)’, written by Lena Pfeilsticker. The method adds up the routing and stock costs for all depots and customers. It is displayed in Figure 38. The routing costs consist of the total costs in former code versions. The stock costs consist of the total demand of every customer multiplied with the stock cost rate lambda.

```
public void calcTotalCosts(CompleteMap map, Input input){
   for( int i = 0; i < map.getSubMaps().size(); i++ )
   {   SubMap aSubMap = map.getSubMaps().get(i);
      Depot aDepot = aSubMap.getDepot();
      stockCosts += aDepot.getBestSolPolicy().getRefillUpToPercent() *
                     aDepot.getExpDemand() * 2 * input.getLambda();
   }
   totalCosts = routingCost + stockCosts;
   }
```

Figure 38 *Method to calculate the total Costs*

The class ‘SubSol’ manages the the sub-maps.

Package Map: The package ‘map’ contains the classes ‘CompleteMap’, ’MapTools’, ’Savings’ and ’SubMap’. It manages the maps and sub-maps of the program. A complete map consists of all built sub-maps and the class ’CompleteMap’ inherits a list of sub-maps as property.

In the class ’MapTools’ a distance matrix of all customers, depots and warehouses is created. Its instance can also be used to find the closest located nodes to all locations on the map

and to calculate the savings for all customers to all depots. The method 'calcDepotSavings(Input input)' enables the user to choose between the calculation of Euclidean distances between map locations or the determination of exact distances via the used parameter 'input'. Then the consecutive approached customers are determined based on the highest savings of the distance matrix. The method 'calcDepotSavings(Input input)' is displayed in Figure 39.

```
public static Savings calcDepotSavings(Input input) {

   if (!input.isEuclideanDistance())
      distanceMatrix = input.getDistances();
   else
      distanceMatrix = distances(input);

   final float[][] savings = savings(input);
   // fill nodes neighbors
   neighbors(input, savings);
   Savings svg = new Savings(distanceMatrix, savings);

   return svg;
   }
```

Figure 39 *Calculating Depot Savings in MapTools*

The class 'Savings' contains methods to calculate savings between different customers and depots.

The class 'SubMap' enables the program to determine the assignment of customers to depots and the demand that needs to be fulfilled for every product per depot.

Package Models: The package 'models' contains the classes 'Customer', 'Delivery', 'Depot', 'ElapsedTime', 'Node', 'Policy', 'SubCustomer', 'Test', 'Vehicle', 'VehType' and 'Warehouse'. It models all items in the given problem. All items contain their assigned attributes as described in the problem description and several methods to modify them. The class 'Depot' was modified by Lena Pfeilsticker to include variables and methods to include inventory levels for every product, to calculate the expected demand, to calculate a product key and a method to calculate the distance to the next warehouse. They are displayed in Figure 40. The expected demand is used to generate a stochastic value for estimating the best inventory policy (see class 'SubAlgorithm' in package 'algorithm').

The method 'calculateProductKey(SubMap subMap) ' cumulates all customer demands assigned to a sub-map, divided into all products, and enables the program to divide the calculated stochastic expected demand into the share of products as it was before the accumulation for the expected demand.

The method 'calculateDistanceToNextWarehouse(Input input, int numDepot)' is used when deciding about which warehouse a depot gets products from for determining the best inventory policy. In the given problem in the present work the number of warehouses is one but the program is aimed to be adaptable to different scenarios and thus this method was implemented.

```
public void calculateExpDemand(SubMap subMap)
{
   expDemand = 0;
     Iterator<Delivery> iter = subMap.getDeliveryNeeds().iterator();
      while (iter.hasNext()){
         Delivery i = (Delivery)iter.next();
         expDemand += i.getCustomer().getTotalDemand();
      }
}

public void calculateProductKey(SubMap subMap){
   productKey = new int[getDiffProducts()];
    Iterator<Delivery> iter = subMap.getDeliveryNeeds().iterator();
      while (iter.hasNext()){
         Delivery m = (Delivery)iter.next();
            for( int i = 1; i <
            m.getCustomer().getProductDemands().length; i++ ){
               productKey[i] += m.getCustomer().getProductDemands()[i];
            }
      }
}

public void calculateDistanceToNextWarehouse(Input input, int numDepot){
   int warehouses = input.getNumWarehouses();
   int depots = input.getNumDepots();
   int nodes = input.getNumNodes();
   float[][] distances = input.getDistances();
   float[] distancesToWarehouses = new float[warehouses];
            for (int j = 0; j < warehouses; j++) {
            distancesToWarehouses [j] =
               distances[numDepot+nodes][j+nodes+depots];
   }
   Arrays.sort(distancesToWarehouses);
   setRoundtripToDepotCosts(distancesToWarehouses[0]);
}
```

Figure 40 *Methods in class 'Depot'*

The class 'ElapsedTime' is used to calculate the time during different methods of the program.

The class 'Policy' was written by Lena Pfeilsticker based on the code of Juan et al. (2014a). It calculates the possible inventory policies and the possibility to compare different ones. Additionally it holds the possibility of setting a current inventory level for the depots. In this work this functionality is not applied but can be used to make a scenario more realistic. This process is displayed in Figure 41.

```
public Policy(float maxCap, float percent, float currentLevel)
{
    refillUpToPercent = percent;
    refillUpToUnits = percent * maxCap;
    unitsToServe = Math.max(0, refillUpToUnits - currentLevel);
   }
```

Figure 41 *Constructor 'Policy'*

The class 'SubCustomer' manages all attributes associated with the sub-customers. They are constructed virtually to simplify the routing process as previously described in the corresponding solving process. For the final solution all sub-customers of the similar locations are re-combined to form a customer again.

The class 'Test' manages the file determining the test(s) to run in the program.

Package Plot: The package 'plot' serves to display the calculated best solution in a graphic and consists of the classes 'InstancePlot' and 'InstancePlot1'.

Package RCWS: The package 'RCWS' is short for randomized CWS heuristic and consists of the classes 'Edge', 'EdgeHelper', 'RandCWS', 'Route' and 'RouteCache'.

In the class 'Edge' the edges are defined and the variable routing costs for one edge can be calculated. One edge is defined as the route connecting two locations on a given map with an origin and an end.

The class 'EdgeHelper' contains several auxiliary methods and serves to help create and improve the solution's routing. It contains methods to

- create edges according to the CWS heuristic, based on computed savings,
- create an edge and its inverse edge with assigned costs,
- create an edge and its inverse edge with costs and calculate its savings with regard to the distance of its assigned depot and
- to complete an edge with its inverse.

The class 'RandCWS' contributes to a great extent to the functionalities and advantages of the code. It provides a biased-randomized calculation of the CWS heuristic. For this purpose an initial 'dummy solution' is created in all sub-maps as previously described in the corresponding solving process. So a return route is built from a depot to all of its assigned customers. Based on this initial solution a biased-randomized CWS heuristic is applied to improve it.

The previously set variables during the creation of the initial dummy solution are reset to create the new biased-randomized solution. The procedure is depicted exemplary in Figure 42. After resetting the variables the edge selection and iterative merging process is performed. When merging routes the conditions to compute a feasible solution are checked. This includes sufficient vehicle capacities for the customer demands and, in case a maximum distance is provided in the input files for the corresponding vehicle type, a check if the maximum constraint is not violated.

```
public class RandCWS
{
…
        /* 1. RESET VARIABLES */
         // dummySol resets isInterior and inRoute in nodes
         SubSol currentSol = generateDummySol(nodes, depotEdges, diffProducts,
           vht);
       …

        /* 2. PERFORM THE EDGE-SELECTION & ROUTING-MERGING ITERATIVE
       PROCESS*/
         …
             // 2.1. Select the next edge from the list (biased-randomized)
              …
             // 2.2. Determine the nodes i < j that define the edge
              …
             // 2.3. Determine the routes associated with each node
              …
             // 2.4. If all necessary conditions are satisfied, merge
             boolean isMergePossible = false;
             isMergePossible = checkMergingConditions(test, iR, jR, ijEdge,
              diffProducts);
              …

        /* 3. RETURN THE SOLUTION */
       …
         // Check if the solution is feasible for the generated dummy routes
         // If so, assign a possible load distribution; else, delete the route
              from the solution.
              …
              if (pending > 0)
                 it.remove(); // this vehicle type can't deliver this dummy
                              route
              else
                 r.setLoadDistribution(ld);
```

Figure 42 *Biased randomized CWS heuristic*

5.3 Output

A complete example for output data is provided in Annex 4. With the final implementation, additional values are displayed when the program finishes. The structure of the program's output is described in the following sections.

Besides the complete map with all routes, the program gives out a text including several data. When first started, the program indicates which file it is getting its data input from and prints out the fractions for the mapping, the routing and the round-robin process. Whenever a new best solution is found it is printed along with the number of solutions found up to this point and the time the program required to compute this solution. An exemplary output is provided in Figure 43.

```
Start solving: testFile Mapping beta: 0.5 CWS beta: 0.5 Round Robin fraction:
1.0 Variance k: 0.5 Stock cost rate Lambda: 0.01
New best found. Cost: 4062.504613876343 among 1 solution in 0.7546984 s
New best found. Cost: 3753.2442858219147 among 2 solutions in 1.421551 s
…
```

Figure 43 *Outputs Part I*

When the given time frame is up, the program prints the number of solutions found and prints the best overall solution with the lowest costs found in this time. This includes the overall costs and demands, the costs, demand and routing of each sub-map, the employed vehicle types and the amount that each customer is served. An exemplary output is provided in Figure 44.

```
Found 4248 solutions in 2700.149 s


Solution [
   Overall solution routing cost=1423.369372844696
   subSolutions=[
   [
      costs=428.8719425201416, demand=[1738, 747, 2997, 2075]
      routes=[
         [

   D20(29,21)>C13(54,17)>C7(39,11)>C6(22,13)>C5(13,11)>C12(12,17)>D20(29,21)

            edges=[[20->13(25.3)], [13->7(16.2)], [7->6(17.1)], [6->5(9.2)],
                   [5->12(6.1)], [12->20(17.5)]]
            costs=165.678635597229
            demand=[0, 0, 2251, 2075]
            vht=VehType [capacity=[5500, 5500]]
          …
         ]
…
```

Figure 44 *Outputs Part II*

The following output was added by Lena Pfeilsticker to demonstrate the new calculated values concerning the stochastic customer demands and the setting of new inventory levels. The output consists of the total costs, the routing costs and the stock costs. For every depot the best refill policy is printed in the order Depot 1, Depot 2, Depot 3. Finally the inventory levels for all products of all depots are printed. The first four numbers indicate the inventory levels for the four products of the first depot in ascending order, then the second, then the third depot.

```
Total costs: 2609.5193667411804
Routing costs: 1423.369372844696
Stock costs: 1186.1499938964844
Best Policies for Depots: [0.5, 0.5, 0.5]

All Inventory Levels: [18083, 10888, 23821, 20432, 23363, 16557, 20437,
13082, 30025, 20176, 19555, 20804]
```

Figure 45 *Outputs Part III*

6 Conclusion

In this work a simheuristic algorithm was presented for solving a rich single-period, multi-depot IRP with stochastic customer demands, stockouts and a heterogeneous fleet. The proposed methodology for the solving process combines several approaches, i.e., simulation, metaheuristics, biased randomization and MS approaches, to enable an integrated decision-making process for the simultaneous planning of inventory levels and routing.

The research area of implementing rich IRPs is challenging because of highly complex constraints and their interdependencies. This is especially the case when introducing stochastic values, such as the customer demand in the present work, because the random behavior increases the number of outcomes exponentially. This leads to long computing times, possibly exceeding the solving ability of the available resources. When aiming at providing a program for real-life IRPs, as in the present work, it is necessary to be able to apply several constraints to the treated IRP due to the complexity of such inventory and routing systems.

In the present work the contribution to the described research area consists of several developments. Due to the combination of several methodologies the developed program is able to deal with realistic IRPs. Its flexibility contributes highly to the adaptability to different scenarios of realistic IRPs and can use real-life data as input for computing feasible, cost-efficient solutions. The flexibility includes the individual setting of vehicle costs, the defining of maximum distances for individual vehicle types, the adding and deleting of customers, depots and warehouses, the possibility of setting current inventory levels etc.

A further significant contribution is that the approach can be used with any probability distribution so the customer demands do not have to follow a normal distribution– which is an unrealistic assumption usually employed in the existing literature.

The program is able to consider different refill policies for every depot in an inventory routing system and computes the lowest cost found in a given time frame by considering the overall routing and inventory costs. The individual refill policies contribute to finding low-cost solutions, compared to other approaches using standard refill policies. The program is also able to consider different stock cost rates.

The simheuristic approach enables to test scenarios with varying variances and thus the evaluation of risks and costs associated with the scenarios. This provides a decision tool for applicants. A complete set of tests has been performed to illustrate the methodology and analyze how costs vary as different uncertainty and costs scenarios are considered.

The overall goal of the present work has been achieved as described previously. During the implementation, details of the program and of the approach became apparent that were not considered before. This led to more possibilities of improving the code that could not be implemented anymore due to the limited time frame.

The further research with the developed program includes, among others, experimenting with different safety stock levels, with time windows or maximum distances and with different probability distributions of customer demands. Also, further research is possible to be considered when supplying the customers before their inventory is completely empty, by installing devices to measure the filling level and then basing an IRP on this assumption.

The overall conclusion is that the program contributes significantly to the research field of IRPs due its flexible applicability and possibility to be used on real-life cases. The program can be used in inventory routing companies for computing low-cost solutions and thus fulfills the need for more research with a realistic framework.

Acknowledgements

This work has been partially supported by the Martin-Schmeißer Foundation.

A special thank you goes to Enoc Martinez and Gabriel Alemany for supporting me during the programming of the code and to Prof. Markus Rabe and Prof. Angel Juan for their advice and given opportunities.

References

Alizadeh, M.; Eskandari, H.; Sajadifar, S.Mehdi (2011): Analyzing a stochastic inventory system for deteriorating items with stochastic lead time using simulation modeling. In Jain, S.; Creasey, R.R.; Himmelspach, J.; White, K.P., et al. (eds.): Proceedings of the 2011 Winter Simulation Conference. Piscataway: IEEE 2011, pp. 1645–1657.

Andradóttir, S. (2006): An overview of simulation optimization via random search. In Henderson, S.G.; Nelson, B.L. (eds.): Handbooks in pperations research and management science, 1. Aufl. s.l.: Elsevier professional 2006, pp. 617–631.

Angelidis, E.; Bohn, D.; Rose, O. (2012): A simulation-based optimization heuristic using self-organization for complex assembly lines. In Laroque, C.; Himmelspach, J.; Pasupathy, R. (eds.): Proceedings of the 2012 Winter Simulation Conference. Berlin: IEEE 2012, pp. 1231–1240.

Anily, S.; Federgruen, A. (1990): One warehouse multiple retailer systems with vehicle routing costs. Management Science 36 (1990) 1, pp. 92–114.

Anily, S.; Federgruen, A. (1993): Two-echelon distribution systems with vehicle routing costs and central inventories. Operations Research 41 (1993) 1, pp. 37–47.

Augerat, P.; Belenguer, J.M.; Benavent, E.; Corberan, A.; Naddef, D. (1998): Separating capacity constraints in the CVRP using tabu search. European Journal of Operational Research 106 (1998) 2, pp. 546–557.

Barnes-Schuster, D.; Bassok, Y. (1997): Direct shipping and the dynamic single-depot/multi-retailer inventory system. European Journal of Operational Research 101 (1997) 3, pp. 509–518.

Bertazzi, L.; Savelsbergh M.; Speranza, M.G. (2008): Inventory routing. In Golden, B.L.; Raghavan, S.; Wasil, E.A. (eds.): The vehicle routing problem. New York, London: Springer 2008, pp. 49–72.

Bramel, J.; Simchi-Levi, D. (1995): A location based heuristic for general routing problems. Operations Research 43 (1995) 4, pp. 649–660.

Caceres, J.; Arias, P.; Guimarans, D.; Riera, D.; Juan, A.A. (2014): Rich vehicle routing problem: survey. ACM Computing Surveys 47 (2014) 2, pp. 1–28.

Campbell, A.; Clarke, L.W.; Savelsbergh M. (2002): Inventory routing in practice. In Toth, P.; Vigo, D. (eds.): The vehicle routing problem. Philadelphia: Society for Industrial and Applied Mathematics 2002, pp. 309–330.

Campbell, A.; Savelsbergh M. (2004a): Delivery volume optimization. Transportation Science 38 (2004a) 2, pp. 210–223.

Campbell, A.; Savelsbergh M. (2004b): Efficient insertion heuristics for vehicle routing and scheduling problems. Transportation Science 38 (2004b) 3, pp. 369–378.

Campbell, A.; Savelsbergh M. (2004c): A decomposition approach for the inventory routing problem. Transportation Science 38 (2004c) 4, pp. 488–502.

Campbell, A.; Savelsbergh M.; Clarke, L.W.; Kleywegt, A. (1998): The inventory routing problem. In Crainic, T.G.; Laporte, G. (eds.): Fleet management and logistics. Boston: Kluwer Academic Publishers 1998, pp. 95–112.

Chao, I.M.; Golden, B.L.; Wasil, E.A. (1993): A new heuristic for the multi-depot vehicle problem that improves upon best-known solution. American Journal of Mathematical and Management Sciences 13 (1993) 3-4, pp. 371–406.

Chien, T.W.; Balakrishnan, A.; Wong, R.T. (1989): An integrated inventory allocation and vehicle routing problem. Transportation Science 23 (1989) 2, pp. 67–76.

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

Coelho, L.C.; Cordeau, J.-F., Laporte, G. (2014): Thirty years of inventory routing. Transportation Science 48 (2014) 1, pp. 1–19.

Cooke, J.A. (1998): VMI: Very mixed impact? Logistics Management and Distribution Report 37 (1998) 12, pp. 51–53.
Croes, G.A. (1985): A method for solving traveling salesman problems. Operations Research 6 (1985) 6, pp. 791–812.
Dantzig, G.; Ramser, J. (1959): The truck dispatching problem. Management Science 6 (1959) 1, pp. 80–91.
Eskandari, H.; Darayi, M.; Geiger, C.D. (2010): Using simulation optimization as a decision support tool for supply chain coordination with contracts. In Johansson, B.; Jain, S.; Montoya-Torres, J.; Hugan, J.; Yücesan, E. (eds.): Proceedings of the 2010 Winter Simulation Conference. Piscataway: IEEE 2010, pp. 1306–1317.
Federgruen, A.; Zipkin, P. (1984): A combined vehicle routing and inventory allocation problem. Operations Research 32 (1984) 5, pp. 1019–1036.
Festa, P.; Resende, M.G.C. (2002): GRASP: An annotated bibliography. In Ribeiro, C.C.; Hansen, P. (eds.): Essays and surveys on metaheuristics. Boston: Kluwer Academic Publishers 2002, pp. 325–367.
Fleurent, C.; Glover, F. (1999): Improved constructive multistart strategies for the quadratic assignment problem using adaptive memory. INFORMS Journal on Computing 11 (1999) 2, pp. 198–204.
Fraza, V. (1998): Streamlining the channel. Industrial Distribution 87 (1998) 9, pp. 73–74.
Gaur, V.; Fisher, M.L. (2004): A periodic inventory routing problem at a supermarket chain. Operations Research 52 (2004) 6, pp. 813–822.
Glover, F. (1977): Heuristics for integer programming using surrogate constraints. Decision Sciences 8 (1977) 1, pp. 156–166.
Glover, F. (1986): Future paths for integer programming and links to artificial intelligence. Computer&Operations Research 13 (1986) 5, pp. 533–549.
Glover, F. (1989): Tabu search. Part I. ORSA Journal on Computing 1 (1989) 3, pp. 190–206.
Glover, F. (2000): Multi-start and strategic oscillation methods - principles to exploit adaptive memory. In Laguna, M.; Gonzalez-Velade, J.L. (eds.): Computing tools for modeling optimization and simulation. Boston: Kluwer Academic Publishers 2000, pp. 1–25.
Godfrey, G.A.; Powell, W.B. (2002): An adaptive dynamic programming algorithm for dynamic fleet management I. Transportation Science 36 (2002) 1, pp. 21–39.
Goel, A.; Gruhn, V. (2005): Solving a dynamic real-life vehicle routing problem. In Haasis, H.D.; Kopfer, H.; Schönberger, J. (eds.): Operations Research Proceedings 2005. Berlin, Heidelberg: Springer 2005, pp. 367–372.
Goel, A.; Gruhn, V. (2008): A general vehicle routing problem. European Journal of Operational Research 191 (2008) 3, pp. 650–660.
Golden, B.L.; Raghavan, S.; Wasil, E.A. (2008): The vehicle routing problem. New York, London: Springer.
Gonzalez, S.; Juan, A.A.; Riera, D.; Elizondo, M.; Fonseca, P. (2012): Sim-RandSHARP: A hybrid algorithm for solving the arc routing problem with stochastic demands. In Laroque, C.; Himmelspach, J.; Pasupathy, R. (eds.): Proceedings of the 2012 Winter Simulation Conference. Berlin: IEEE 2012, pp. 1–11.
Hoos, H.H.; Stützle, T. (2005): Stochastic local search: foundations and applications. San Francisco: Morgan Kaufmann Publishers.
Jaillet, P.; Huang, L.; Bard, M.; Dror, M. (2002): Delivery cost approximations for inventory routing problems in a rolling horizon framework. Transport Science 36 (2002) 3, pp. 292 – 300.
Juan, A.A.; Faulin, J.; Jorba, J.; Riera, D.; Masip, D.; Barrios, B. (2010a): On the use of Monte Carlo simulation, cache and splitting techniques to improve the Clarke and Wright savings heuristics. Journal of the Operational Research Society 62 (2010a) 6, pp. 1085–1097.
Juan, A.A.; Faulin, J.; Ruiz, R.; Barrios, B.; Caballé, S. (2010b): The SR-GCWS hybrid algorithm for solving the capacitated vehicle routing problem. Applied Soft Computing 10 (2010b) 1, pp. 215–224.

Juan, A.A.; Faulin, J.; Ferrer, A.; Lourenço, H.R.; Barrios, B. (2013): MIRHA. Multi-start biased randomization of heuristics with adaptive local search for solving non-smooth routing problems. TOP 21 (2013) 1, pp. 109–132.

Juan, A.A.; Grasman, S.E.; Caceres-Cruz, J.; Bektaş, T. (2014a): A simheuristic algorithm for the single-period stochastic inventory-routing problem with stock-outs. Simulation Modelling Practice and Theory 46 (2014a), pp. 40–52.

Juan, A.A.; Pascual, I.; Guimarans, D.; Barrios, B. (2014b): Combining biased randomization with iterated local search for solving the multidepot vehicle routing problem. International Transactions in Operational Research 21 (2014b), pp. 1–21.

Juan, A.A.; Faulin, J.; Grasman, S.E.; Rabe, M.; Figueira, G. (2015): A review of Simheuristics: extending metaheuristics to deal with stochastic optimization problems. Operations Research Perspectives 2 (2015), pp. 62–72.

Lahyani, R.; Khemakhem, M.; Semet, F. (2015): Rich vehicle routing problems. European Journal of Operational Research 241 (2015) 1, pp. 1–14.

Laroque, C.; Klaas, A.; Fischer, J.H.; Kuntye, M. (2012): Fast converging, automated experiment runs for material flow simulations using distributed computing and combined metaheuristics. In Laroque, C.; Himmelspach, J.; Pasupathy, R. (eds.): Proceedings of the 2012 Winter Simulation Conference. Berlin: IEEE 2012, pp. 102–111.

Lourenço, H.R.; Martin, O.; Stützle, T. (2001): A beginner's introduction to iterated local search. In Bartz-Beielstein, T. (eds.): Proceedings of the 4th Metaheuristics International Conference. Berlin, New York: Springer 2001, pp. 1–11.

Lourenço, H.R.; Martin, O.; Stützle, T. (2002): Iterated Local Search. In Glover, F.; Kochenberger, G. (eds.): Handbook of metaheuristics. Norwell: Kluwer Academic Publishers 2002, pp. 321–353.

Martí, R.; Resende, M.G.C.; Ribeiro, C.C. (2013): Multi-start methods for combinatorial optimization. European Journal of Operational Research 226 (2013) 1, pp. 1–8.

Martin, O.; Otto, S.W.; Felten, E.W. (1992): Large-step Markov chains for the TSP incorporating local search heuristics. Operations Research Letters 11 (1992) 4, pp. 219–224.

Mjirda, A.; Jarboui, B.; Macedo, R.; Hanafi, S.; Mladenovic, N. (2014): A two phase variable neighborhood search for the multi-product inventory routing problem. Computer&Operations Research 52 (2014) Part B, pp. 291–299.

Pillac, V.; Gendreau, M.; Guéret, C.; Medaglia, A.L. (2013): A review of dynamic vehicle routing problems. European Journal of Operational Research 225 (2013) 1, pp. 1–11.

Reiman, M.; Rubio, R.; Wein, L.M. (1999): Heavy traffic analysis of the dynamic stochastic inventory-routing problem. Transportation Science 33 (1999) 4, pp. 361–372.

Schmid, V.; Doerner, K.F.; Laporte, G. (2013): Rich routing problems arising in supply chain management. European Journal of Operational Research, 224 (2013), 3, pp. 435–448.

Silberschatz, A.; Galvin, P.Baer; Gagne, G. (2010): Operating system concepts, 8. ed., internat. student version. Hoboken: John Wiley & Sons.

Suhl, L.; Mellouli, T. (2009): Optimierungssysteme, 2., überarb. Aufl. Dordrecht: Springer.

Talbi, E.-G. (2009): Metaheuristics. From design to implementation. Hoboken: John Wiley & Sons.

Tang, L.; Wang, X. (2006): Iterated local search algorithm based on very large-scale neighborhood for prize-collecting vehicle routing problem. The International Journal of Advanced Manufacturing Technology 29 (2006) 11-12, pp. 1246–1258.

Toth, P.; Vigo, D. (2002): The vehicle routing problem. Philadelphia: Society for Industrial and Applied Mathematics.

Trudeau, P.; Dror, M. (1992): Stochastic inventory routing route design with stockouts and route failures. Transportation Science 26 (1992) 3, pp. 171–184.

van Dijk, Nico M.; van der Sluis, Erik (2008): Practical optimization by OR and simulation. Simulation Modelling Practice and Theory 16 (2008) 8, pp. 1113–1122.

Waller, M.; Johnson, M.Eric; Davis, T. (1999): Vendor-managed inventory in the retail supply chain. Journal of business logistics 20 (1999) 1, pp. 183–204.

Yu, Y.; Chu, F.; Chen, H. (2006): A model and algorithm for large scale stochastic inventory routing problem. In Chu, C.; Kacem, I. (eds.): Proceedings of Service Systems and Service Management International Conference. Piscataway: IEEE 2006, pp. 355–360.

List of Figures

List of Tables

List of Abbreviations

COP	Combinatorial Optimization Problem
CVRP	Capacitated Vehicle Routing Problem
CWS	Clarke and Wright Savings (Heuristic)
HVRP	Heterogeneous Fleet Vehicle Routing Problem
ILS	Iterated Local Search
IRP	Inventory Routing Problem
MS	Multi-Start
NP(-hard)	Non-Deterministic Polynomial-time (-hard)
PDP	Pick-up and Delivery
RNG	Random Number Generator
RVRP	Rich Vehicle Routing Problem
TSP	Travelling Salesman Problem
VMI	Vendor Managed Inventory
VRP	Vehicle Routing Problem
VRPTW	Vehicle Routing Problem with Time Windows
WH	Warehouse

Annex

Annex 1

Customer ID	X-Coordinate	Y-Coordinate	Demand Product 1	Demand Product 2
1	37	52	5	5
2	49	49	7	7
3	52	64	22	22
4	20	26	34	34
5	40	30	44	44
6	21	47	12	12
7	17	63	23	23
8	31	62	42	42
9	52	33	32	32
10	51	21	11	11
11	42	41	34	34
12	31	32	12	12
13	5	25	12	12
14	12	42	13	13
15	36	16	14	14
16	52	41	15	15
17	27	23	6	6
18	17	33	8	8
19	13	13	33	33
20	50	32	1	1

Annex 2

Customer ID	X-Coordinate	Y-Coordinate	Demand Product 1	Demand Product 2	Demand Product 3
1	37	52	5	5	5
2	49	49	7	7	7
3	52	64	22	22	22
4	20	26	34	34	34
5	40	30	44	44	44
6	21	47	12	12	12
7	17	63	23	23	23
8	31	62	42	42	42
9	52	33	32	32	32
10	51	21	11	11	11
11	42	41	34	34	34
12	31	32	12	12	12
13	5	25	12	12	12

14	12	42	13	13	13
15	36	16	14	14	14
16	52	41	15	15	15
17	27	23	6	6	6
18	17	33	8	8	8
19	13	13	33	33	33
20	50	32	1	1	1
21	37	32	5	5	5
22	49	52	7	7	7
23	52	49	22	22	22
24	20	64	34	34	34
25	40	26	44	44	44
26	21	30	12	12	12
27	17	47	23	23	23
28	31	63	42	42	42
29	52	62	32	32	32
30	55	33	11	11	11
31	42	21	34	34	34
32	31	41	12	12	12
33	5	32	12	12	12
34	12	25	13	13	13
35	36	42	14	14	14
36	52	16	15	15	15
37	27	41	6	6	6
38	17	23	8	8	8
39	13	33	33	33	33
40	50	13	1	1	1
41	37	49	5	5	5
42	49	64	7	7	7
43	52	26	22	22	22
44	20	30	34	34	34
45	40	47	44	44	44
46	21	63	12	12	12
47	17	62	23	23	23
48	31	33	42	42	42
49	52	21	32	32	32
50	51	41	11	11	11
51	42	32	34	34	34
52	31	25	12	12	12
53	5	42	12	12	12
54	12	16	13	13	13
55	36	41	14	14	14
56	52	23	15	15	15
57	27	33	6	6	6
58	17	13	8	8	8
59	13	32	33	33	33
60	50	52	1	1	1
61	37	64	5	5	5

62	49	26	7	7	7
63	52	30	22	22	22
64	20	47	34	34	34
65	40	63	44	44	44
66	21	62	12	12	12
67	17	33	23	23	23
68	31	21	42	42	42
69	52	41	32	32	32
70	47	35	11	11	11
71	42	25	34	34	34
72	31	42	12	12	12
73	5	16	12	12	12
74	12	41	13	13	13
75	36	23	14	14	14
76	60	33	15	15	15
77	27	13	6	6	6
78	17	32	8	8	8
79	13	52	33	33	33
80	50	45	1	1	1
81	52	58	5	5	5
82	20	40	7	7	7
83	40	60	22	22	22
84	21	20	34	34	34
85	17	38	44	44	44
86	31	47	12	12	12
87	60	63	23	23	23
88	45	60	42	42	42
89	42	33	32	32	32
90	31	21	11	11	11
91	5	49	34	34	34
92	12	32	12	12	12
93	36	25	12	12	12
94	61	42	13	13	13
95	27	16	14	14	14
96	17	41	15	15	15
97	13	23	6	6	6
98	50	36	8	8	8
99	37	13	33	33	33
100	49	12	1	1	1

Annex 3

Customer ID	X-Coordinate	Y-Coordinate	Demand Product 1	Demand Product 2	Demand Product 3	Demand Product 4
1	61	9	892	329	0	337
2	71	9	656	201	588	542

3	80	9	589	681	704	922
4	90	8	0	912	912	291
5	162	6	761	210	732	318
6	13	11	530	294	196	693
7	22	13	314	128	785	132
8	39	11	297	138	860	0
9	147	13	586	293	242	423
10	53	12	746	402	300	806
11	161	11	782	615	789	539
12	173	13	820	418	350	0
13	12	17	595	109	279	806
14	54	17	0	0	145	431
15	72	14	476	505	194	442
16	81	17	689	816	586	104
17	89	18	682	458	15	0
18	94	16	233	193	0	523
19	166	15	70	653	596	0
20	18	19	0	104	746	0
21	41	21	638	0	349	0
22	61	18	555	711	404	22
23	170	19	802	426	781	577
24	9	24	706	0	532	71
25	53	25	856	877	953	57
26	82	24	363	266	0	634
27	19	27	155	927	10	562
28	38	28	511	0	727	239
29	43	29	0	0	771	169
30	59	30	183	109	656	557
31	66	27	523	364	369	346
32	97	30	22	128	532	683
33	161	26	386	60	208	857
34	8	30	417	366	795	585
35	29	30	120	137	20	859
36	75	34	770	866	505	125
37	146	34	138	102	788	166
38	16	36	612	905	493	637
39	40	36	961	419	544	0
40	86	38	90	217	0	724
41	97	36	312	941	355	47
42	168	34	0	536	220	356
43	9	42	433	325	0	890
44	24	38	749	112	180	0
45	36	41	826	664	711	714
46	47	39	604	222	0	495
47	72	39	730	105	566	281
48	82	41	914	58	751	177
49	149	41	8	436	497	115
50	151	40	15	14	469	0

51	160	39	220	107	737	554
52	9	44	396	390	718	821
53	92	45	0	872	797	578
54	14	47	183	936	0	909
55	45	49	682	294	931	912
56	57	48	850	568	385	208
57	68	49	857	694	673	527
58	166	46	273	384	492	681
59	22	53	502	546	737	150
60	36	52	517	558	223	8
61	65	53	593	80	412	258
62	75	54	98	369	901	582
63	86	52	694	319	364	212
64	97	50	523	0	0	596
65	147	52	586	843	751	901
66	50	54	373	587	336	217
67	155	56	211	779	0	255
68	12	62	165	295	604	50
69	27	59	13	20	783	940
70	35	59	164	759	364	538
71	44	60	257	0	352	158
72	83	59	953	228	446	0
73	90	61	714	363	730	611
74	19	62	0	580	825	916
75	78	64	425	56	0	183
76	96	62	806	521	937	200
77	149	62	29	248	703	295
78	159	66	766	0	412	796
79	10	68	652	140	571	150
80	26	68	399	814	893	9
81	41	68	891	889	461	147
82	52	69	395	386	269	306
83	64	67	481	261	194	0
84	74	68	410	0	54	0
85	170	70	900	441	100	820
86	16	72	268	803	498	362
87	56	74	72	435	589	449
88	86	72	85	80	318	748
89	146	71	223	425	715	646
90	153	72	865	221	0	525
91	10	75	884	128	832	76
92	26	75	684	933	219	395
93	35	75	503	660	830	269
94	50	74	819	202	187	244
95	64	74	602	0	754	701
96	91	77	689	841	329	418
97	159	77	236	0	406	115
98	25	79	0	951	73	900

99	45	81	731	642	0	0
100	73	81	335	744	289	840

Annex 4

```
Start solving: test11 Mapping beta: 0.9 CWS beta: 0.7 Round Robin fraction:
1.0
New best found. Cost: 3097.763965816719 among 1 solutions in 0.21153577 s
New best found. Cost: 2771.234927548427 among 7 solutions in 0.67912984 s
New best found. Cost: 2754.5637371490257 among 27 solutions in 2.2724779 s
New best found. Cost: 2627.4937833666236 among 520 solutions in 17.633913 s
New best found. Cost: 2613.7401683892426 among 3158 solutions in 90.01653 s
New best found. Cost: 2554.301313027424 among 6614 solutions in 186.1463 s
New best found. Cost: 2548.262887036315 among 15540 solutions in 431.71088 s
New best found. Cost: 2544.607862815759 among 36833 solutions in 1010.6004 s
New best found. Cost: 2508.3683721926836 among 53606 solutions in 2579.0735 s
Found 57514 solutions in 3604.8608 s


Solution [
   Overall solution cost=2508.3683721926836
   subSolutions=[
   [
      id=552771, costs=628.3200100963842, demand=[12069, 8913, 12147, 9676]
      routes=[
         [
            id=30545936
            edges=[[D100(29,21)->C44(36,41)], [C44(36,41)->C58(22,53)],
[C58(22,53)->C67(12,62)], [C67(12,62)->C51(9,44)], [C51(9,44)->C37(16,36)],
[C37(16,36)->D100(29,21)]]
            costs=101.81019970698509
            demand=[1603, 1864, 3263, 0]
            vht=Vehicle [vehType=VehType [capacity=[1800, 3150, 4050]]], Load
Distribution=[0, 1, 2]
            deliveries=[[44<-2], [44<-0], [44<-1], [58<-2], [67<-1], [67<-0],
[67<-2], [51<-2], [37<-1], [37<-0], [37<-2]]
         ],
         [
            id=30545875
            edges=[[D100(29,21)->C6(22,13)], [C6(22,13)->C5(13,11)],
[C5(13,11)->C12(12,17)], [C12(12,17)->C23(9,24)], [C23(9,24)->C26(19,27)],
[C26(19,27)->D100(29,21)]]
            costs=55.65043620479082
            demand=[1550, 0, 1513, 2264]
            vht=Vehicle [vehType=VehType [capacity=[1800, 3150, 4050]]], Load
Distribution=[0, 2, 3]
            deliveries=[[6<-3], [6<-0], [6<-2], [5<-3], [5<-0], [5<-2], [12<-
3], [23<-0], [23<-2], [23<-3], [26<-3]]
         ],
         [
            id=30545908
            edges=[[D100(29,21)->C34(29,30)], [C34(29,30)->C43(24,38)],
[C43(24,38)->C38(40,36)], [C38(40,36)->C24(53,25)], [C24(53,25)-
>D100(29,21)]]
            costs=75.91893311577297
            demand=[2566, 1545, 1677, 0]
```

```
            vht=Vehicle [vehType=VehType [capacity=[1800, 3150, 4050]]], Load
Distribution=[1, 2, 0]
            deliveries=[[34<-1], [43<-2], [43<-0], [43<-1], [38<-2], [38<-0],
[38<-1], [24<-2], [24<-0], [24<-1]]
         ],
         [
            id=45678
            edges=[[D100(29,21)->C34(29,30)], [C34(29,30)->C44(36,41)],
[C44(36,41)->C27(38,28)], [C27(38,28)->C28(43,29)], [C28(43,29)->C24(53,25)],
[C24(53,25)->C13(54,17)], [C13(54,17)->C21(61,18)], [C21(61,18)->C1(71,9)],
[C1(71,9)->C0(61,9)], [C0(61,9)->D100(29,21)]]
            costs=123.82366496474086
            demand=[511, 0, 1643, 3370]
            vht=Vehicle [vehType=VehType [capacity=[1800, 3150, 4050]]], Load
Distribution=[0, 2, 3]
            deliveries=[[34<-3], [27<-3], [27<-0], [27<-2], [44<-3], [28<-3],
[28<-2], [24<-3], [13<-3], [13<-2], [21<-3], [1<-3], [0<-3]]
         ],
         [
            id=30545876
            edges=[[D100(29,21)->C6(22,13)], [C6(22,13)->C19(18,19)],
[C19(18,19)->C5(13,11)], [C5(13,11)->C12(12,17)], [C12(12,17)->C33(8,30)],
[C33(8,30)->C26(19,27)], [C26(19,27)->D100(29,21)]]
            costs=70.02312057543487
            demand=[1167, 1928, 1830, 0]
            vht=Vehicle [vehType=VehType [capacity=[1800, 3150, 4050]]], Load
Distribution=[0, 1, 2]
            deliveries=[[6<-1], [19<-2], [19<-1], [5<-1], [12<-1], [12<-0],
[12<-2], [33<-1], [33<-0], [33<-2], [26<-1], [26<-0], [26<-2]]
         ],
         [
            id=30545911
            edges=[[D100(29,21)->C20(41,21)], [C20(41,21)->C21(61,18)],
[C21(61,18)->C1(71,9)], [C1(71,9)->C0(61,9)], [C0(61,9)->C7(39,11)],
[C7(39,11)->D100(29,21)]]
            costs=91.91023012133587
            demand=[3038, 1379, 2201, 0]
            vht=Vehicle [vehType=VehType [capacity=[1800, 3150, 4050]]], Load
Distribution=[1, 2, 0]
            deliveries=[[20<-2], [20<-0], [21<-2], [21<-0], [21<-1], [1<-2],
[1<-0], [1<-1], [0<-0], [0<-1], [7<-1], [7<-0], [7<-2]]
         ],
         [
            id=30545906
            edges=[[D100(29,21)->C37(16,36)], [C37(16,36)->C58(22,53)],
[C58(22,53)->C67(12,62)], [C67(12,62)->C53(14,47)], [C53(14,47)->C51(9,44)],
[C51(9,44)->C42(9,42)], [C42(9,42)->C33(8,30)], [C33(8,30)->D100(29,21)]]
            costs=109.18342540732374
            demand=[1514, 2197, 0, 4042]
            vht=Vehicle [vehType=VehType [capacity=[1800, 3150, 4050]]], Load
Distribution=[0, 1, 3]
            deliveries=[[37<-3], [58<-3], [58<-0], [58<-1], [67<-3], [53<-1],
[53<-0], [53<-3], [51<-3], [51<-0], [51<-1], [42<-0], [42<-1], [42<-3], [33<-
3]]
         ]]
   ],
   [
      id=552774, costs=880.375114408449, demand=[10367, 8804, 12318, 11346]
      routes=[
```

```
         [
            id=30545980
            edges=[[D101(154,19)->C22(170,19)], [C22(170,19)->C41(168,34)],
[C41(168,34)->C32(161,26)], [C32(161,26)->C31(97,30)], [C31(97,30)-
>C15(81,17)], [C15(81,17)->C3(90,8)], [C3(90,8)->C8(147,13)], [C8(147,13)-
>D101(154,19)]]
            costs=205.6696425811007
            demand=[0, 590, 2250, 2504]
            vht=Vehicle [vehType=VehType [capacity=[1800, 3150, 4050]]], Load
Distribution=[1, 2, 3]
            deliveries=[[22<-3], [41<-1], [41<-3], [41<-2], [32<-3], [31<-2],
[15<-2], [3<-1], [3<-2], [3<-3], [8<-3]]
         ],
         [
            id=30545977
            edges=[[D101(154,19)->C32(161,26)], [C32(161,26)->C57(166,46)],
[C57(166,46)->C84(170,70)], [C84(170,70)->C88(146,71)], [C88(146,71)-
>C76(149,62)], [C76(149,62)->C64(147,52)], [C64(147,52)->C48(149,41)],
[C48(149,41)->C49(151,40)], [C49(151,40)->D101(154,19)]]
            costs=133.18138079310734
            demand=[1805, 1760, 3935, 0]
            vht=Vehicle [vehType=VehType [capacity=[1800, 3150, 4050]]], Load
Distribution=[1, 0, 2]
            deliveries=[[32<-2], [32<-0], [32<-1], [57<-2], [57<-0], [57<-1],
[84<-2], [84<-0], [84<-1], [88<-1], [88<-0], [88<-2], [76<-2], [64<-2], [48<-
1], [48<-0], [48<-2], [49<-1], [49<-0], [49<-2]]
         ],
         [
            id=3369284
            edges=[[D101(154,19)->C10(161,11)], [C10(161,11)->C4(162,6)],
[C4(162,6)->C9(153,12)], [C9(153,12)->C8(147,13)], [C8(147,13)-
>D101(154,19)]]
            costs=41.848126140310505
            demand=[2875, 1520, 242, 0]
            vht=Vehicle [vehType=VehType [capacity=[1800, 3150, 4050]]], Load
Distribution=[1, 2, 0]
            deliveries=[[10<-1], [4<-1], [4<-0], [10<-0], [9<-1], [9<-0],
[8<-0], [8<-1], [8<-2]]
         ],
         [
            id=30546003
            edges=[[D101(154,19)->C36(146,34)], [C36(146,34)->C48(149,41)],
[C48(149,41)->C64(147,52)], [C64(147,52)->C76(149,62)], [C76(149,62)-
>C88(146,71)], [C88(146,71)->C89(153,72)], [C89(153,72)->C66(155,56)],
[C66(155,56)->D101(154,19)]]
            costs=115.69007935615963
            demand=[1691, 2091, 0, 2903]
            vht=Vehicle [vehType=VehType [capacity=[1800, 3150, 4050]]], Load
Distribution=[0, 1, 3]
            deliveries=[[36<-3], [48<-3], [64<-3], [64<-0], [64<-1], [76<-3],
[76<-0], [76<-1], [88<-3], [89<-1], [89<-0], [89<-3], [66<-1], [66<-0], [66<-
3]]
         ],
         [
            id=30545963
            edges=[[D101(154,19)->C18(166,15)], [C18(166,15)->C11(173,13)],
[C11(173,13)->C22(170,19)], [C22(170,19)->C50(160,39)], [C50(160,39)-
>C36(146,34)], [C36(146,34)->D101(154,19)]]
            costs=80.86417298476981
```

```
            demand=[2050, 1706, 3252, 0]
            vht=Vehicle [vehType=VehType [capacity=[1800, 3150, 4050]]], Load
Distribution=[1, 0, 2]
            deliveries=[[18<-2], [18<-0], [18<-1], [11<-2], [11<-0], [11<-1],
[22<-1], [22<-0], [22<-2], [50<-2], [50<-0], [50<-1], [36<-1], [36<-0], [36<-
2]]
         ],
         [
            id=30545981
            edges=[[D101(154,19)->C10(161,11)], [C10(161,11)->C4(162,6)],
[C4(162,6)->C9(153,12)], [C9(153,12)->C17(94,16)], [C17(94,16)->C15(81,17)],
[C15(81,17)->C31(97,30)], [C31(97,30)->D101(154,19)]]
            costs=177.38689095390305
            demand=[2451, 1749, 0, 2973]
            vht=Vehicle [vehType=VehType [capacity=[1800, 3150, 4050]]], Load
Distribution=[1, 3, 0]
            deliveries=[[10<-3], [4<-1], [4<-0], [4<-3], [9<-1], [9<-0], [9<-
3], [17<-3], [17<-0], [17<-1], [15<-1], [15<-0], [15<-3], [31<-1], [31<-0],
[31<-3]]
         ],
         [
            id=30545999
            edges=[[D101(154,19)->C50(160,39)], [C50(160,39)->C57(166,46)],
[C57(166,46)->C84(170,70)], [C84(170,70)->C96(159,77)], [C96(159,77)-
>C77(159,66)], [C77(159,66)->D101(154,19)]]
            costs=125.73482159909805
            demand=[1002, 0, 818, 2966]
            vht=Vehicle [vehType=VehType [capacity=[1800, 3150, 4050]]], Load
Distribution=[0, 2, 3]
            deliveries=[[50<-3], [57<-3], [84<-3], [96<-0], [96<-2], [96<-3],
[77<-3], [77<-0], [77<-2]]
         ]]
   ],
   [
      id=552777, costs=999.6732476878504, demand=[23655, 20569, 20934, 18408]
      routes=[
         [
            id=30546074
            edges=[[D102(64,62)->C94(64,74)], [C94(64,74)->C86(56,74)],
[C86(56,74)->C93(50,74)], [C93(50,74)->C81(52,69)], [C81(52,69)->C70(44,60)],
[C70(44,60)->D102(64,62)]]
            costs=63.52651062816858
            demand=[2073, 0, 1562, 1858]
            vht=Vehicle [vehType=VehType [capacity=[6000, 10500, 13500]]],
Load Distribution=[0, 2, 3]
            deliveries=[[94<-3], [94<-0], [94<-2], [86<-3], [93<-3], [93<-0],
[93<-2], [81<-2], [81<-0], [81<-3], [70<-2], [70<-0], [70<-3]]
         ],
         [
            id=30119923
            edges=[[D102(64,62)->C61(75,54)], [C61(75,54)->C52(92,45)],
[C52(92,45)->C56(68,49)], [C56(68,49)->C60(65,53)], [C60(65,53)->C65(50,54)],
[C65(50,54)->C54(45,49)], [C54(45,49)->C59(36,52)], [C59(36,52)->C69(35,59)],
[C69(35,59)->C68(27,59)], [C68(27,59)->C73(19,62)], [C73(19,62)->C97(25,79)],
[C97(25,79)->C92(35,75)], [C92(35,75)->C93(50,74)], [C93(50,74)->C81(52,69)],
[C81(52,69)->D102(64,62)]]
            costs=180.48316458607235
            demand=[0, 2403, 2368, 6668]
```

```
            vht=Vehicle [vehType=VehType [capacity=[6000, 10500, 13500]]],
Load Distribution=[1, 2, 3]
            deliveries=[[61<-3], [52<-1], [52<-2], [52<-3], [56<-2], [60<-3],
[65<-3], [54<-3], [59<-3], [69<-3], [68<-3], [73<-2], [73<-1], [73<-3], [97<-
2], [97<-1], [97<-3], [92<-3], [93<-3], [81<-3]]
         ],
         [
            id=30546060
            edges=[[D102(64,62)->C74(78,64)], [C74(78,64)->C99(73,81)],
[C99(73,81)->C87(86,72)], [C87(86,72)->C95(91,77)], [C95(91,77)->C75(96,62)],
[C75(96,62)->C72(90,61)], [C72(90,61)->C62(86,52)], [C62(86,52)->C63(97,50)],
[C63(97,50)->C40(97,36)], [C40(97,36)->C39(86,38)], [C39(86,38)->C47(82,41)],
[C47(82,41)->C46(72,39)], [C46(72,39)->C35(75,34)], [C35(75,34)->C25(82,24)],
[C25(82,24)->C2(80,9)], [C2(80,9)->C14(72,14)], [C14(72,14)->C30(66,27)],
[C30(66,27)->C29(59,30)], [C29(59,30)->C45(47,39)], [C45(47,39)->C79(26,68)],
[C79(26,68)->C78(10,68)], [C78(10,68)->C90(10,75)], [C90(10,75)->C85(16,72)],
[C85(16,72)->C91(26,75)], [C91(26,75)->C80(41,68)], [C80(41,68)->C55(57,48)],
[C55(57,48)->C60(65,53)], [C60(65,53)->D102(64,62)]]
            costs=354.19253967894326
            demand=[7683, 4694, 0, 10163]
            vht=Vehicle [vehType=VehType [capacity=[6000, 10500, 13500]]],
Load Distribution=[1, 3, 0]
            deliveries=[[74<-3], [74<-0], [74<-1], [99<-3], [99<-0], [99<-1],
[87<-3], [87<-0], [87<-1], [95<-3], [72<-3], [75<-3], [63<-0], [63<-3], [62<-
3], [46<-3], [46<-0], [46<-1], [35<-3], [47<-3], [47<-0], [47<-1], [39<-3],
[39<-0], [39<-1], [40<-3], [40<-0], [40<-1], [25<-1], [25<-3], [25<-0], [2<-
3], [14<-3], [30<-0], [30<-1], [30<-3], [29<-3], [45<-1], [45<-0], [45<-3],
[79<-3], [78<-3], [78<-0], [78<-1], [90<-3], [85<-3], [91<-1], [91<-0], [91<-
3], [80<-3], [55<-3], [55<-0], [55<-1], [60<-0], [60<-3]]
         ],
         [
            id=226886
            edges=[[D102(64,62)->C82(64,67)], [C82(64,67)->C86(56,74)],
[C86(56,74)->C98(45,81)], [C98(45,81)->C80(41,68)], [C80(41,68)->C92(35,75)],
[C92(35,75)->C91(26,75)], [C91(26,75)->C85(16,72)], [C85(16,72)->C90(10,75)],
[C90(10,75)->C78(10,68)], [C78(10,68)->C79(26,68)], [C79(26,68)->C68(27,59)],
[C68(27,59)->C69(35,59)], [C69(35,59)->C59(36,52)], [C59(36,52)->C54(45,49)],
[C54(45,49)->C65(50,54)], [C65(50,54)->D102(64,62)]]
            costs=157.44694526954882
            demand=[5841, 7360, 7501, 0]
            vht=Vehicle [vehType=VehType [capacity=[6000, 10500, 13500]]],
Load Distribution=[0, 1, 2]
            deliveries=[[82<-2], [82<-0], [82<-1], [86<-2], [80<-2], [80<-0],
[80<-1], [98<-1], [98<-0], [92<-2], [92<-0], [92<-1], [91<-2], [91<-0], [91<-
1], [85<-2], [85<-0], [85<-1], [90<-2], [78<-1], [78<-0], [78<-2], [79<-1],
[79<-0], [79<-2], [68<-1], [68<-0], [68<-2], [69<-1], [69<-0], [69<-2], [59<-
1], [54<-1], [54<-0], [54<-2], [65<-1], [65<-0], [65<-2]]
         ],
         [
            id=30446368
            edges=[[D102(64,62)->C83(74,68)], [C83(74,68)->C99(73,81)],
[C99(73,81)->C87(86,72)], [C87(86,72)->C95(91,77)], [C95(91,77)->C75(96,62)],
[C75(96,62)->C72(90,61)], [C72(90,61)->C71(83,59)], [C71(83,59)->C62(86,52)],
[C62(86,52)->C61(75,54)], [C61(75,54)->C56(68,49)], [C56(68,49)->C46(72,39)],
[C46(72,39)->C35(75,34)], [C35(75,34)->C47(82,41)], [C47(82,41)->C40(97,36)],
[C40(97,36)->C16(89,18)], [C16(89,18)->C2(80,9)], [C2(80,9)->C14(72,14)],
[C14(72,14)->C30(66,27)], [C30(66,27)->C29(59,30)], [C29(59,30)->C55(57,48)],
[C55(57,48)->D102(64,62)]]
            costs=244.02408752511747
```

```
            demand=[9532, 5624, 9541, 0]
            vht=Vehicle [vehType=VehType [capacity=[6000, 10500, 13500]]],
Load Distribution=[1, 2, 0]
            deliveries=[[83<-0], [83<-2], [99<-2], [99<-0], [99<-1], [87<-2],
[87<-0], [87<-1], [95<-0], [95<-2], [75<-1], [75<-0], [75<-2], [72<-2], [62<-
2], [62<-0], [62<-1], [71<-1], [71<-0], [71<-2], [61<-2], [61<-0], [61<-1],
[56<-2], [56<-0], [56<-1], [46<-2], [46<-0], [46<-1], [35<-1], [35<-0], [35<-
2], [47<-2], [47<-0], [47<-1], [40<-2], [16<-2], [16<-0], [16<-1], [2<-2],
[14<-2], [14<-0], [14<-1], [30<-2], [29<-1], [29<-0], [29<-2], [55<-1], [55<-
0], [55<-2]]
         ]]
   ]]
]
```

Annex 5

Customer ID	X-Coordinate	Y-Coordinate	Demand Product 1	Demand Product 2	Demand Product 3	Demand Product 4
1	61	9	892	329	0	337
2	71	9	656	201	588	542
3	80	9	589	681	704	922
4	90	8	0	912	912	291
5	162	6	761	210	732	318
6	13	11	530	294	196	693
7	22	13	314	128	785	132
8	39	11	297	138	860	0
9	147	13	586	293	242	423
10	53	12	746	402	300	806
11	161	11	782	615	789	539
12	173	13	820	418	350	0
13	12	17	595	109	279	806
14	54	17	0	0	145	431
15	72	14	476	505	194	442
16	81	17	689	816	586	104
17	89	18	682	458	15	0
18	94	16	233	193	0	523
19	166	15	70	653	596	0
20	18	19	0	104	746	0

Eidesstattliche Versicherung

_____ _____

Name, Vorname Matr.-Nr.

Ich versichere hiermit an Eides statt, dass ich die vorliegende Bachelorarbeit/Masterarbeit* mit dem Titel

_____

_____

_____

selbstständig und ohne unzulässige fremde Hilfe erbracht habe. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie wörtliche und sinngemäße Zitate kenntlich gemacht. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

_____ _____

Ort, Datum Unterschrift

*Nichtzutreffendes bitte streichen

Belehrung:

Wer vorsätzlich gegen eine die Täuschung über Prüfungsleistungen betreffende Regelung einer Hochschulprüfungsordnung verstößt, handelt ordnungswidrig. Die Ordnungswidrigkeit kann mit einer Geldbuße von bis zu 50.000,00 € geahndet werden. Zuständige Verwaltungsbehörde für die Verfolgung und Ahndung von Ordnungswidrigkeiten ist der Kanzler/die Kanzlerin der Technischen Universität Dortmund. Im Falle eines mehrfachen oder sonstigen schwerwiegenden Täuschungsversuches kann der Prüfling zudem exmatrikuliert werden. (§ 63 Abs. 5 Hochschulgesetz - HG -)

Die Abgabe einer falschen Versicherung an Eides statt wird mit Freiheitsstrafe bis zu 3 Jahren oder mit Geldstrafe bestraft.

Die Technische Universität Dortmund wird gfls. elektronische Vergleichswerkzeuge (wie z.B. die Software „turnitin“) zur Überprüfung von Ordnungswidrigkeiten in Prüfungsverfahren nutzen.

Die oben stehende Belehrung habe ich zur Kenntnis genommen:

_____ _____

Ort, Datum Unterschrift