ERASMUS UNIVERSITY ROTTERDAM
Erasmus School of Economics

Master Thesis Econometrics and Management Science
Operations Research and Quantitative Logistics

The flower seller problem

Author:
Dion den Heijer
431470

Supervisor:
Dr. T.A.B. Dollevoet
Second assessor:
Dr. D. Galindo Pecin

July 8, 2020

Abstract

In this thesis the flower seller problem is introduced. The problem concerns the delivery of a range of commodities to a number of customers for which demands and revenues are known. The supply of each commodity is also known. When the salesman arrives at a customer he has to fulfill the demand of the customer for every commodity as far as he is able to. For some commodities, the total demand is larger than the supply and thus the route that the salesman travels has impact on the profit he generates. Three exact methods are introduced to optimize the route which the salesman has to take in order the maximize his profits. These methods can generate optimal values for instances involving up to 15 customers and 20 commodities or up to 30 customers and 2 commodities. For larger instances a heuristic is used to return good routes in reasonable time. We expect this heuristic to return very good routes for the instances tested. The heuristic consistently outperforms naive heuristics that can be applied by a real-life salesman.

Contents

1 Introduction

The traveling salesman problem (TSP) is one of the most widely studied problems in combinatorial optimization. In the TSP a salesman travels from a starting city to a set of other cities and back to the starting city in the shortest route possible (Dantzig et al., 1954). For this problem, numerous extensions have been developed in the literature. An example is the traveling salesman problem with profits (TSP with profits) as described by Feillet, Dejax and Gendreau (2005), in which customers provide the salesman with profits, and not all customers have to be visited.

A real-life application of a traveling salesman problem with profits comes in the form of a salesman that travels to his customers and offers them a range of products to buy. For example a flower seller who fills up his truck with roses, tulips and sunflowers and makes his route along his customers, selling his products. Each customer has a certain demand and wants to pay a certain price for every different type of flower. The flower seller wants to make his tour as profitable as possible, considering the revenue he collects from selling his products as well as the traveling costs he makes by choosing a specific route. He wants to sell certain flowers at certain customers because he knows they pay a high price for these products. In other words the flower seller needs to pick the best order in which he visits his customers. This real-life problem resembles the TSP with profits. When considering this problem we make some assumptions: We assume that the supply for each product is fixed and the sum of the supplies over all products is equal to the capacity of the vehicle. Moreover, we assume that the demands and revenues for all products are fixed and given for all customers. Finally we assume that not going to a customer is very costly for customer relations and should be avoided at all times. This applies to arriving at a customer with no inventory left for any commodity as well. For this reason we assume that the flower seller has purchased enough units of at least one product to satisfy the demand of all his customers for this commodity.

The problem the flower seller faces differs somewhat from the TSP with profits in the literature. First, in the TSP with profits, only a subset of customers are visited, whereas in this problem the salesman is required to visit all customers. Whenever the salesman arrives at a customer he operates under a "can not say no" principle in which he fulfills the demand for all different commodities this customer has, as long as he is able to. The problem then is not selecting which customers to visit as in the TSP with profits, but selecting which customers to visit first, since the salesman does not want to be out of stock when arriving at a customer who pays a high price for this commodity. The second difference lies in the fact that although the flower seller always starts and ends at the same location which is the depot, his last customer is always set. This customer has no specified demand and buys up the left over inventory for all commodities at a price that gives the lowest profits among all customers. We call this customer the unloader. By adding this additional piece of information the problem becomes finding a most efficient $s - t$ path visiting all customers. The source s in this case is the depot, the sink t is the unloader. Since the salesman is required to visit every customer, but can only visit them exactly once, this problem becomes finding a Hamiltonian path that maximizes the profits.

It is clear that if the total demand of all commodities is below the supply for that commodity, this problem becomes a regular Eucledian Hamiltonian path problem. On the other hand, if the demand for all commodities for all customers is larger than the supply for this commodity, the salesman might arrive at customers with all his inventory already sold. Therefore, we assume that there is one "control" commodity for which the salesman has more supply than demand, for all other commodities the demand will be higher than the supply. This way each customer can be served for at least one commodity. With this assumption, the optimal route no longer has to be the shortest route but will now depend on which customers are visited first.

The problem the flower seller faces will be named the flower seller problem (FSP). The aim of this thesis is to find a solution method that provides good routes in reasonable time. These routes will be compared to naive routes that a salesman could apply in real-life scenarios. The FSP, which is an extension of the TSP with profits in which elements of revenue management come into play, is to the best of our knowledge not researched in the literature and seems novel. Aside from the scientific relevance, the problem is based on a real-world problem and could therefore offer relevant insights for practical applications.

To find an optimal path we develop a formulation for the FSP. We try to use this formulation in a branch-and-bound algorithm to solve different instances involving different combinations of number of customers and commodities, starting with very small instances. In order to solve these instances, we use the commercial solver CPLEX. We also develop a dynamic programming approach as well as a label correcting algorithm. We compare these methods with each other in order to find which exact method is most suited to find optimal values for the FSP.

We can reduce all instances of the FSP to an instance of a Euclidean Hamiltonian path problem by setting all demands for all commodities to zero. Since finding a Euclidean Hamiltonian path is known to be NP-complete (Altinel et al., 2000), we can assume the FSP to be NP-complete as well. Therefore, it is not reasonable to expect the exact approaches to generate optimal paths for larger instances within reasonable time. In order to find good routes for larger instances, we propose 2 local search heuristic algorithms. In both algorithms random routes are improved until a local optimum is found. This step is repeated for a fixed time frame and the best route is chosen. We want to compare these heuristics with each other as well as make a combination of the two and find how well they generate good routes.

The remainder of this thesis is as follows: in section 2 we present a literature review on known problems which are related to the FSP. In section 3 we provide the problem description and the mathematical formulation. In section 4 we describe the exact methods and heuristics that we want to use. Section 5 describes the manner in which the data is generated. In section 6 we present the results and in section 7 we give our conclusion and provide a discussion.

2 Literature review

In this section we provide an overview of the literature that is relevant to our problem. We consider the FSP to be an extension of the TSP with elements of TSPs with profits, the Hamiltonian path problem, the elementary shortest path problem with resource constraints and revenue management. We provide a short overview of the literature on these subjects along with some methods that are used to approach these problems that seem relevant for our problem.

2.1 Traveling salesman problem

Consider a graph $G = G(V, E)$, where V is a set of vertices and E is a set of edges. The TSP consists of finding a minimum distance circuit passing trough each vertex in V exactly once. Traveling between vertices i and j comes at a certain cost c_{ij}. The TSP can be symmetrical, where $c_{ij} = c_{ji}$ for all $i, j \in V$, or asymmetrical. The cost matrix C is said to satisfy the triangle inequality if $c_{ij} + c_{jk} \geq c_{ik}$ for all $i, j, k \in V$. This occurs in Euclidean problems. It can be proven that the TSP is NP-hard by transforming the Hamiltonian circuit (HC) problem, which is known to be NP-complete (Laporte, 1992). There are many solution methods developed for the TSP. Some of the most well-known are listed below.

2.1.1 Exact methods

The exact methods provide us with an optimal solution for the TSP. Since the TSP is NP-hard these methods tend to be fast for small instances but the running time starts to increase drastically when the instances get larger. The main exact solution approaches are branch-and-bound and dynammic programming.

Branch-and-bound A number of integer linear programming formulations have been proposed for the TSP. These include the well known formulation of Dantzig, Fulkerson and Johnson (DFJ) (Dantzig et al., 1954) and the formulation of Miller, Tucker and Zemlin (MTZ) (1960). These formulations are used in branch-and-bound algorithms.

Dynamic programming Held and Karp have proposed dynamic programming approaches for a number of sequencing problems including the TSP (Held and Karp, 1962). They introduce DP-variable $C(S, l)$ which is the minimum cost route visiting all nodes in set S and ending at node l. Let's assume without loss of generality that the tour starts and ends at node 1. Then we can initialize the DP by taking $S = v$ for every node v and set $C(\{v\}, v) = c_{1,v}$, where $c_{1,v}$ is the cost of traveling from 1 to v. The recursive relation is as follows

$$C(S, v) = \min_{u \in S \backslash \{v\}} \{C(S \backslash \{v\}, u) + c_{u,v}\}$$

where u is the last node to be visited before node v. If set S is the set of all nodes, the optimal solution of the TSP is given by

$$C(S, 1) = \min_{v \in S \backslash \{1\}} \{C(S \backslash \{1\}, v) + c_{v,1}\}$$

The complexity of this algorithm is $\mathcal{O}(n^2 2^n)$.

2.1.2 Tour construction algorithms

Tour construction algorithms are used to construct a tour, which does not need to be optimal. This tour can be improved later on by tour improvement algorithms. Some examples of tour construction algorithm are listed below.

Nearest neighbour The nearest neighbour tour construction algorithm constructs a tour by selecting a random vertex as a starting point and traveling to the nearest unvisited vertex until all vertices are visited (Bellmore and Nemhauser, 1968).

Insertion methods Insertion methods are a class of methods in which, by a sequence of steps, tours are constructed for a progressively larger subset of vertices. We start with an initial subtour and add vertices in a specific manner. For example, we want to add a vertex to the subtour T. We have two choices to make: which vertex to add and how to change the tour in order to add this vertex. For the first choice we can find the vertex that is closest to T meaning that for all vertices that are not in T we find the shortest distance of this vertex to the nearest vertex in T and select vertex v for which this distance is minimal. When we have selected v we can add it by changing the subtour such that the total distance is minimized (Rosenkrantz et al., 1977).

2.1.3 Tour improvement algorithms

These algorithms take an existing tour and try to improve it. An example is the $k - opt$ algorithm: Replace k edges of the the tour with k different edges in a way that shortens the tour length. The simplest version of this algorithm is $2 - opt$ in which 2 edges in the tour are exchanged. Lin and Kernighan (1973) have used $k - opt$ to create one of the most effective heuristic algorithms for solving the symmetric TSP (Helsgaun, 2000).

The Lin-Kernighan algorithm Lin and Kernighan (1973) argued that fixing k in a $k-opt$ algorithm is a serious drawback, since computational efforts rise rapidly with increasing k and it is not clear what k to use as the best compromise between quality and running time. They propose an algorithm in which k is dynamically adjusted in every iteration. In every iteration they search for the k most "out-of-place" pairs of vertices. Their algorithm contains the following steps:

1. Generate an initial solution.
2. Select the most "out-of-place" pair (which maximizes improvement). If more gain can be made by selecting additional pairs, repeat this step until no more pairs can be selected.
3. Exchange the k pairs found in step 2. If the solution improves, go to step 2, otherwise go to step 4.
4. Repeat from step 1 with a different initial solution if desired.

Tour improvement algorithm for the asymmetric traveling salesman problem When the cost of traveling from i to j are not necessarily the same as from j to i the TSP is called Asymmetric (ATSP). Kanellakis and Papadimitriou (1980) have devised a local search algorithm based on the Lin-Kernighan algorithm. They only consider uneven k and choose the acrs they want to add and remove in a sequential manner. Their local search heuristic performs well and they claim that the probability of finding the optimum becomes very high when applying this heuristic for up to 100 cities. For the 3-opt heuristic they consider all different possibilities as they found that this substantially improved the probability of finding the optimum at a modest increase in computation time. For higher values of k they only consider a certain set of candidates.

2.2 Hamiltonian path problem

The Euclidean Hamiltonian path problem (HPP) is closely related to the TSP and can be defined as follows: given a set of nodes V, a source and sink node, s and t respectively and the distances for each pair of nodes, find the shortest path starting at s and ending at t. The HPP can be transformed into a TSP by adding a distance of $-\infty$ between s and t (Altinel et al., 2000). Therefore, most solutions methods to the HPP utilise the underlying solution methods of the TSP.

2.3 Traveling salesman problem with profits

The TSP with profits is a collection of problems gathered by Feillet et al. (2005). This collection of problems proposes to select customers based on a profit value that is gained when a visit occurs. The general objective of these problems is to maximize the profits while minimizing costs. We will discuss the different types of TSPs with profits and the most common approaches that are used to solve these problems.

2.3.1 Different types of traveling salesman problems with profits

The TSPs with profits consist of three generic problems, differing in the way the two objectives are addressed. The first type is where the cost objective is stated as a constraint and the objective is to maximize the profits. In the second type, the profit amount is a constraint and the objective is to minimize the travel cost. The third way is the problem where both objective are in the objective function.

The cost objective is stated as a constraint The aim of this problem is to find a route that maximizes the profit such that travel cost do not exceed a certain maximal value. While this problem is commonly known as the orienteering problem (OP), it can also be found under different names. Laporte and Martello (1990) dubbed it the selective TSP and Kataoka and Morito (1988) call this the maximum collection problem.

The profit amount is a constraint Here the objective becomes to find a route that minimizes the traveling cost while gathering at least a certain profit value. It is most commonly

known as the prize collecting TSP (PCTSP) as defined originally by Balas (1989). It has also been named the quota TSP (Awerbuch et al., 1998).

Both objectives are in the objective function The aim is to find a tour that minimizes the costs minus the profits. This problem has been defined by Dell'Amico et al. (1995) as the profitable tour problem (PTP) This problem has rarely been studied as such in the literature. Instead it is often encountered as a sub problem in column generation algorithms for a number of different routing problems. In these sub problems, the profit values are the duals which are subtracted at each vertex. In most cases, the sub problem is solved as an elementary shortest path problem between two copies of the depot (Feillet et al., 2005).

2.3.2 Exact solution approaches

The exact solution approaches for the TSP with profits are often branch-and-bound methods adapted from the procedures created for the TSP. There are two general approaches. The first is relaxing the subtour elimination constraints, which works well for asymmetric situations. The second approach is relaxing the constraint that enforces every node to have a single successor. This relaxation can be defined as a shortest spanning 1-arborescence problem (Feillet et al., 2005).

2.3.3 Classical heuristics

The main heuristic approach is improving an existing solution by using one of the following four main operations: Adding a vertex to a route, deleting a vertex from a route, resequencing the route and replacing a vertex of the route with a vertex outside the route. Since the TSP with profits has two objectives these operations usually lead to improvement in one objective at the expense of the other. Many heuristic procedures are obtained from a combination of these four operations. The following procedures are described by Feillet et al. (2005).

Greedy insertion This is a procedure of iteratively inserting vertices in to the solution based on the highest profit value or lowest traveling cost. This insertion takes place until it is either no longer possible due to the traveling cost exceeding a constraint (in case of the OP), a certain profit value is obtained (for the PCTSP) or no improvement is possible (in the PTP).

Greedy deletion This procedure starts off with an initial route that is a solution of the TSP and iterativelty deletes vertices from this solution. The vertex to be deleted is chosen in such a way that this deletion will improve the objective. For the PTP insertion and deletion can be applied simultaneously.

Path-extension procedure This procedure involves extending a path until no more vertices can be added. It is faster but less effective than the greedy insertion procedures. Because of their efficiency, a randomized behavior can be introduced and the most promising vertices can be chosen in a probabilistic fashion as in Tsiligirides (1984).

Sweep-based procedure Also proposed by Tsiligrides (1984), this procedure is based on the sweeping algorithm for the TSP, introduced by Wren and Holliday (1972). In this procedure, the geographic area is divided into sectors determined by two circles and an arc of given length. Routes are built up within these sectors and the sectors are subject to change by rotating the arcs or changing the radius of a circle. This algorithm is executed many times and the best route is selected. For similar computing times, this procedure provides inferior results to the path-extension procedures.

Partitioning-based procedure This method is developed by Chao et al. (1996). The procedure does not focus on a single route. Unlike the before mentioned procedures, multiple feasible routes are considered and the best route is improved. Vertices are partitioned in a set of feasible routes. From this set the best route is considered. After the partition, local search procedures are used to move vertices between routes in order to improve the best route in the set.

2.3.4 Meta heuristics

The four main operations described in the beginning of section 2.3.3 can be very effective, provided that a solution is not trapped in a local minimum or in a cycle. Metaheuristics can provide a solution that bypasses these problems. There are several types of metaheuristics applied to TSPs with profits as described in Feilet et al. (2005).

Tabu search Tabu search based procedures developed by Ramesh and Brown (1991) as well as a by Gendrau et al. (1998) emerge among the most efficient approaches to solving the OP. To avoid cycling in the local search scheme, a deleted vertex is assigned a tabu status for a randomly selected number of periods, meaning that this vertex cannot be inserted in these periods.

Deterministic annealing This solution procedure, developed by Chao et al. (1996), uses a partitioning-based procedure to generate a set of routes. In order to create the routes, a nearest neighbour insertion procedure is used to add vertices to a route. Then local search procedures are applied. It is allowed to accept solutions that have a worse objective value, as long as this deterioration is not larger than a given percentage of the current solution value. The probability of accepting a solution that has a worse objective value lessens over time.

2.4 The elementary shortest path problem with resource constraints

The shortest path problem is one of the most well studied topics in graph theory. The problem is defined as follows: given an instance containing a set of vertices V, a source vertex s, a sink vertex t with $s, t \in V$ and a set of weighted edges E, over the set V, find the path between s and t which has the minimum weight (Madkour et al., 2017).

An extension to the shortest path problem comes in the form of the elementary shortest path problem (ESSP). In this problem, every node can be visited only once. The ESSP can be extended even further if the nodes have a certain consumption of limited resources. The

problem then becomes to find a shortest path that does not exceed these resource limits. This is known as the elementary shortest path problem with resource constraints (ESPPRC). This problem is often encountered as a sub problem in a column generation scheme for different vehicle routing problems (VRP) such as the VRP with time windows and the capacitated VRP (Feillet et al., 2004).

The ESPPRC is known to be NP-hard in the strong sense (Dror, 1994). A recent survey by Pugliese and Guerriero (2013) states that the ESPPRC was first addressed by Beasley and Christofides (1989) who defined a mathematical formulation as well as a dynamic programming formulation. The dynamic programming formulation is the same as described by Aneja et al. (1983) for the shortest path problem with resource constraints (SPPRC). To address the elementary aspect, Beasley and Christofides suggest using a dummy resource bounded by one that represents the inclusion of a node in a path. The authors claimed that the dynamic programming formulation could not be efficient since adding the dummy resource would make the state space very large (Pugliese and Guerriero, 2013).

For the SPPRC, a label correcting reaching algorithm was first proposed by Desrochers (1988). This algorithm was an extension of the Ford-Bellman algorithm, taking the resource constraints into account. Each node receives a number of labels throughout the algorithm which stand for partial paths. Nodes are iteratively treated until no new labels can be created. To limit the number of labels, dominance rules are applied (Feillet et al., 2004). This algorithm was adapted for the ESPPRC context by Feillet et al. (2004) by using the dummy resource as proposed by Beasley and Christofides (1989). They also introduced the concept of unreachable vertices. Here, a label cannot be extended to a vertex if the resource consumption of the label plus the resource consumption of this vertex exceeded the capacity. They implement the amount of unreachable nodes in the dominance rules and manage to speed up the label correcting algorithm considerably. The algorithm was further improved by Righini and Salani (2008), who used bi-directional dynamic programming and decremental state space relaxation. In bi-directional dynamic programming, paths are created from the source as well as the sink, these paths are joined in the final step. When applying resource based bounding, this process greatly reduces the number of states generated. Resource based bounding can be done on capacity, where forward and backward labels are no longer extended when their resource consumption value is more than half of the total capacity. Decremental state space relaxation is an iterative algorithm. In the first iteration, the elementary aspect is completely relaxed allowing for fast computation of an optimal path. If this path contains duplicate vertices, these vertices are added to a critical vertices set. For this set the constraint holds that these vertices can only be visited once. Every iteration this set is extended, if duplicate vertices in a path are found. If no duplicate vertices are found, the optimal value is obtained.

2.5 Revenue management

The objective of revenue management, also known as yield management is to maximize profits by offering different fares for the same product (McGill and Van Ryzin, 1999). The FSP contains elements of revenue management since the goal is to maximize profits by

selecting which customers to visit first which resembles offering a different fare for a selection of commodities.

3 Problem formulation

In this section, a mathematical formulation for the FSP is presented. The problem is formulated as a minimum cost flow problem with additional constraints. The objective is to maximize the total profit. We define the profit as the total revenue made by selling commodities at customers minus the traveling cost. The additional constraints are tied to the resources and make sure that the salesman sells the demand at each customer if he is able to, otherwise he will sell how much he has left in stock. First the sets, parameters and variables will be specified, after which the formulation is presented.

Sets

N The set of customers

P The set of products

Parameters

q_p Total capacity of product p

d_{ip} Demand of customer i for product p

a_{ip} Revenue of customer i for product p

c_{ij} Cost of going from customer i to customer j

Variables

Binary variables

x_{ij} Binary variable indicating if customer j is visited directly after customer i

z_{ip} Binary variable indicating if the demand at customer i is higher than the amount of product p left in stock. (If this variable equals 1, the (full) demand of customer i can not be satisfied)

Continuous variables

y_{ip} The amount of product p bought by customer i

δ_{ip} The amount of product p left in stock after leaving customer i

Mathematical formulation

Below the mathematical formulation is presented. For this formulation we consider a graph $G = (V, E)$, where V represents the customers and E is the set of edges representing the road from one customer to another. Let S be a set of nodes and $\Pi(S)$ be a set of edges with one end in S and one end not in S. The supply q_p is given for all commodities $p \in P$. Note that we assume that every customer can be visited from every other customer, therefore G is a complete graph. We are looking for the shortest elementary path from the source s, to the sink t.

$$\max \sum_{i \in N} \sum_{p \in P} a_{ip} y_{ip} - \sum_{(i,j) \in E} c_{ij} x_{ij} \tag{1}$$

$$\text{s.t} \sum_{e \in \Pi(s)} x_e = 1 \tag{2}$$

$$\sum_{e \in \Pi(t)} x_e = 1 \tag{3}$$

$$\sum_{e \in \Pi(i)} x_e = 2, \qquad \forall i \in N \backslash \{s, t\} \tag{4}$$

$$\delta_{ip} - \delta_{jp} \geq d_{jp} - M(1 - x_{ij}) - M z_{jp} \qquad \forall p \in P, \forall (i,j) \in E \tag{5}$$

$$\delta_{jp} \leq M(1 - z_{jp}) \qquad \forall p \in P, \forall j \in N \tag{6}$$

$$y_{jp} \leq \delta_{ip} + (1 - x_{ij})M \qquad \forall p \in P, \forall j \in N \tag{7}$$

$$y_{jp} \leq d_{jp} \qquad \forall p \in P, \forall j \in N \tag{8}$$

$$\sum_{(i,j) \in S} x_{ij} \leq |S| - 1 \qquad \forall S \subseteq V \backslash \{s, t\}, |S| \geq 2 \tag{9}$$

$$x_{ij} \in \{0, 1\} \qquad \forall (i,j) \in E \tag{10}$$

$$z_{ip} \in \{0, 1\} \qquad \forall i \in N, \forall p \in P \tag{11}$$

$$0 \leq y_{ip} \leq q_p \qquad \forall i \in N, \forall p \in P \tag{12}$$

$$0 \leq \delta_{ip} \leq q_p \qquad \forall i \in N, \forall p \in P \tag{13}$$

In this formulation the goal is to obtain the maximum profit (1). Constraints (2) - (4) are the flow constraints and the elementary constraints. They make sure that every node has a predecessor (except for s, which is the depot) and every node has a successor (except node t, which is the node representing the unloader). Constraints (5) - (7) are big M constraints where M is a large number. Constraints (5) state that the difference in the amount of commodity left when leaving node i and when leaving node j should be greater than the demand at node j for this commodity if j is directly visited after i. This holds unless the demand at j is larger than the current supply for this commodity, in which case this sets the variable z_{jp} to 1. Constraints (6) are always satisfied as long as the demand at j for product p is less than or equal to the amount left in stock. If this is not the case, the remainder of the product is sold at customer j and when the truck leaves, the stock of this product is zero. Constraints (7) make sure that the amount of product p that is bought at customer j can not exceed the amount of inventory left of this product after leaving customer i, if j is visited directly after i. Constraints (8) make sure that the amount of product p that is bought

by customer j does not exceed the demand. Constraints (9) are the subtour elimination constraints from the DFJ formulation. Constraints (10) - (13) ensure that the variables are binary or continuous and that the amount of commodity p being sold, or the amount left, after visiting customer i can never be higher than the supply for this commodity. In order to make sure that the unloader buys up the rest of the inventory for all commodities, his demands for every commodity are set to the total starting supply of that commodity.

4 Methodology

In this section, the methods that are used to create a route for the FSP are explained in more detail. We first describe the exact methods and then we explain how we build the heuristics.

4.1 Exact methods

We will use three different exact methods to find the optimal objective. We will use a branch-and-bound method, a dynamic programming approach and a label correcting algorithm.

4.1.1 Branch-and-bound

We will use the mathematical formulation as described in section 3 and the commercial solver CPLEX to find an optimal solution. The subtour elimination constraints lead to an exponential number of constraints. To reduce the number of constraints needed we dynamically add subtour elimination constraints each time the model returns a best solution. If in the current solution, one or more subtours are present, we add subtour eliminination constraints for the vertices that are in these subtours and the problem is solved once more until the best solution contains no more subtours, then this solution is optimal. Given the complexity of the problem we expect the run time to explode when increasing the size of the instances.

4.1.2 Dynamic programming

We construct a DP algorithm based on Held and Karp (1962). We introduce the DP variable $C(S, v)$ as the most profitable route ending at customer v having visited all customers in set S. Let $p_{u,v}$ be the profit (revenue minus cost) generated by traveling from u to v. Every route starts at the depot and thus the node s associated with this depot has to be in every subset S. We use a binary power set method to generate all customer subsets. This method uses binary values in order to indicate whether a node is present in a certain subset. For example if we have 2 customers, 00 would indicate an empty subset, 01 or 10 would indicate that customer 1 or product 2 is the only customer in the subset and 11 would indicate that both customers are present in the subset. After creating all subsets, we remove all sets that do not contain node s or that contain node t (which represents the unloader) when the size of the set is not equal to the number of customers (since node t has to be visited last). We initialize all $C(S, v)$ where $|S| = 2$ by taking the route from the depot to v ($v \neq t$). The recurrent relation is:

$$C(S, v) = \max_{u \in S \setminus \{v\}} \{C(S \setminus \{v\}, u) + p_{u,v}\}$$

where u is the last node visited before v. Since the route always ends at node t the optimal solution is found by:

$$C(V, v) = \max_{u \in V \setminus \{t\}} \{C(V \setminus \{t\}, u) + p_{u,t}\}$$

4.1.3 Label correcting algorithm

We will also implement the label correcting algorithm as described by Feillet et al. (2004) and modify it for multiple resources. The algorithm will run for $n-1$ iterations, where n is the number of vertices. In every iteration the labels are extended and dominance rules are applied. To save memory after every iteration all previous labels are deleted since we only need to consider all labels that contain a full path from s to t. After $n-1$ iterations, the vertices only contain labels that describe a path from s to that node going trough all vertices except t. t is added last and the minimum cost path among all labels is retrieved. Since the paths are always of length n we again expect the run time to explode for larger instances.

A label at node v contains the route up until v, the total profit earned by taking this route, how much stock we have left for every commodity and the iteration in which the label is generated. At every iteration we extend every label from its current v to every unvisited node of this label, with the exception of node t. Since there are $(n-1)!$ different possible routes from s to t (because the n'th customer is always given) we apply dominance rules in order to prevent generating $(n-1)!$ labels. Consider the labels l and l^* on node u and the set of N customers and P commodities. V_i is a binary value indicating whether vertex i is included in the path of this label and R is the total profit of the label. Then l^* dominates l if and only if:

$$V_i = V_i^*, \ \forall i \in N$$
$$R < R^*$$

In other words it must hold that every node visited by l has also been visited by l^* and that the total profit of l^* is at higher than the profit of l.

4.2 Heuristics

We start by constructing two heuristics which can easily be seen as tour construction principles a real-life salesman would apply. Namely a tour based on the shortest distance to each customer and a tour based on the highest profit that can be gathered from each customer with the current inventory.

In order to generate a good route, we will apply two tour improvement heuristics. The first is based on the similarity of customers and the second is a $3-opt$ algorithm. Both heuristics consist of two steps. In the first step a random route is generated and in the second step this route is iteratively improved until a local maximum is found. Both heuristics are then run for a pre-specified amount of time and the best route is saved.

4.2.1 Greedy distance heuristic

Starting at the depot, this heuristic will select the node that is closest to the current location and travel there. This step is repeated until all vertices are visited, finishing at t. This method is similar to the nearest neighbour heuristic discussed in section 2.1.3.

4.2.2 Greedy profit heuristic

Starting at the depot, this heuristic will select the node that adds the most profit by multiplying the minimum of the demand and the current stock for all commodities with their respective profits and subtract the traveling cost from this amount. This step is repeated until all vertices are visited, finishing at t.

4.2.3 Random generated path

From a uniform distribution we select random vertices to be added to the path. This step is repeated until all vertices are visited, finishing at t.

4.2.4 Similarity heuristic

The idea of this heuristic is to swap two customers in a route based on the amount of each commodity these customers have bought. The idea is swap two customers in a manner that has little impact on the rest of the route. This can be done by comparing the amount that is bought by the first customer with the demand of the second customer for all commodities. If these two values are similar, the amount left in the inventory after leaving the new customer would be roughly the same as with the old customer. If we swap a customer whose profits are low with a similar customer whose profits are high, swapping these customers would most likely result in an improved route.

In order to use this heuristic we calculate a dissimilarity value for every pair of nodes not including the depot and the unloader. The first node in the pair n_1 always precedes the second node n_2. The amount of commodity p sold at n_1 is the minimum of d_{n_1p} and the stock of this commodity directly before arriving at n_1 denoted as I_{n_1}. The amount of commodity p that could be sold when swapping these nodes is the minimum of d_{n_2p} and I_{n_1}. The absolute difference between these values is summed up for all commodities and the total is the dissimilarity value of these two nodes.

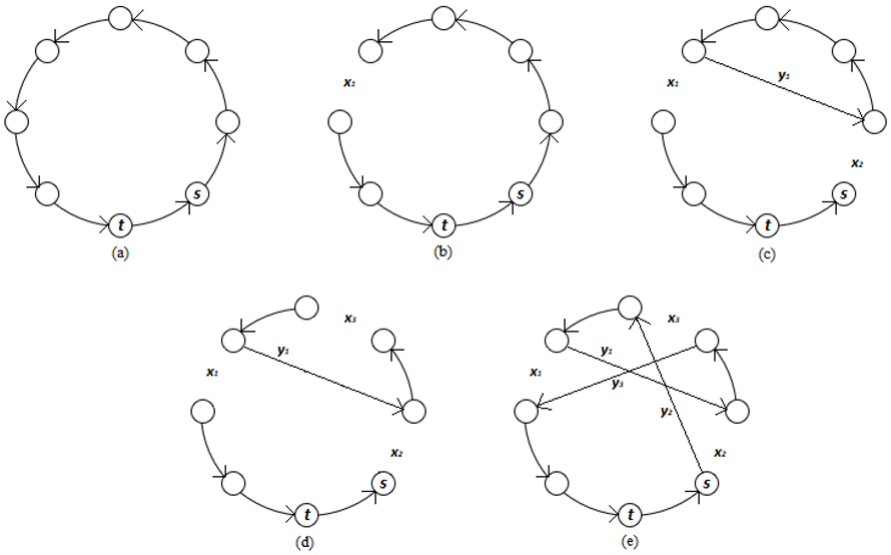
In every iteration we find the minimum dissimilarity pair of nodes which improves the objective value. If no two nodes can be swapped in order to improve the objective value the algorithm terminates. This heuristic is named H1. In order to find a good route we use H1 for 60 seconds and return the route giving the highest profit. If a known optimum is found before the time limit, the algorithm is terminated and the runtime required to find this optimum is stored.

4.2.5 3-Opt heuristic

This heuristic is based on Kanellakis and Papadimitriou (1980). Let a solution to the FSP consist of n nodes, which represent the customers and $n-1$ arcs which represent the order in which the customers are visited. In this algorithm an extra arc is added between the unloader and the depot. In this manner an initial tour is formed as is shown in Figure 1a. In this 3-opt algorithm the original tour is interrupted by taking away one arc x_1 (1b). The tail of x_1 is then connected to a different node in the tour by the new arc y_1. The original arc x_2 that went towards this node is also removed from the tour (1c). These steps divide the original tour in a new subtour and a path. The next step is to break the subtour and

connect the path. This is done by removing a third arc x_3 within the tour (1d), connecting the head of x_3 with the end of the path by inserting arc y_2 and connecting the tail of x_3 with the start of the path by inserting arc y_3 (1e).

Figure 1: The 3-opt heuristic



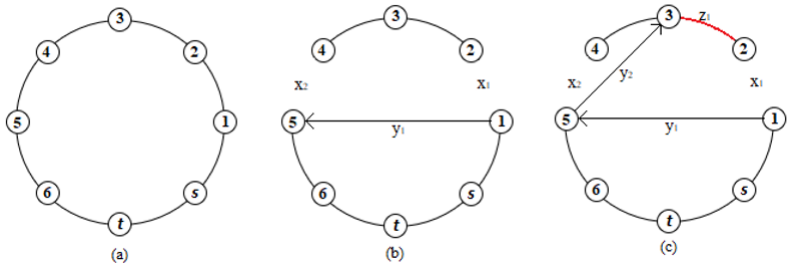
As is the case in Kanellakis and Papadimitriou (1980), we try all different combinations of x_1, y_1 and y_2 (Note that x_2, x_3 and y_3 are uniquely determined by the choices of x_1, y_1 and y_2). We presort the set of options for x_1 based on the amount of profit that is obtained from traversing the arcs in the current tour, starting with the least profitable arc. We then sort the values of y_1 based on the amount of profit they generate, starting with the highest value. For every combination of x_1 and y_1 we find the most profitable y_2 and complete the 3-opt. We then calculate the profit of the new tour. The tour which has the highest amount of profit for all these options is saved. This heuristic is named H2. We use H2 in the same manner as H1 in order to find good routes.

4.2.6 Combining H1 and H2

We can also use a combination of H1 and H2, where we first let H2 run for 60 seconds and then apply H1 to the best route since it is not possible for some pairs of nodes to be swapped in H2, for an example see Figure 2. The route as in 2a is s - 1 - 2 - 3 - 4 - 5 - 6 - t. Suppose we want to swap nodes 2 and 5 in order to get route s - 1 - 5 - 3 - 4 - 2 - 6 - t. In H2 this would be done by choosing x_1, x_2 and y_1 as in 2b. But in order to get the desired route we need to choose y_2 as in 2c. But this would require the route part 2 - 3 - 4 to be changed, by removing z_1. This is not possible in H2, but nodes 2 and 5 can easily be swapped in H1.

We combine H1 and H2 as follows: we let H2 run for 60 seconds, then we apply H1 to the best route generated in H2. When no improvement is found, the algorithm is terminated. If H1 finds an improved route we apply H2 to this route and see if we can improve even further. If this is not the case the algorithm is terminated. If we can improve this route we again

Figure 2: Swapping 2 customers in the 3-opt heuristic



apply H1 and the cycle repeats for as long as an improved route is found. The combination of H1 and H2 heuristics is named H3.

4.3 Upper bounds

Since there are no comparable problems, we cannot find instances for which we can compare a known optimum value to our heuristics. It is also unreasonable to expect optimal values for the exact methods, especially for the larger instances. We thus need to create upper bounds to compare our heuristics.

4.3.1 Maximum profits and minimal cost

We can create an upper bound by taking the maximum profit that can be obtained from selling all commodities and subtracting the cost of the shortest $s-t$ path. Thus the commodity is always sold at the best available price until there is no more demand or no more supply. For this upper bound we maximize the profits and minimize the cost separately. Therefore, this value is the maximum profit that could theoretically be obtained by the salesman. However, due to the nature of the problem it is highly improbable that the optimal value is very close to this upper bound. Therefore, we search for tighter upper bounds by trying to optimize a subset of commodities.

4.3.2 Optimizing a subset of commodities

For instances involving a large number of products we decide to create a subset of products. We then try to solve the problem involving only this subset of products to optimallity. We then add the commodities that are not in the subset by taking the maximum profit that can be obtained by selling these commodities. In order to create these tighter upper bounds we try three different subsets.

Two least demanded commodities This subset contains the two commodities (not including the control commodity) for which the total demand is the closest to their supply. The reason for selecting this subset is as follows: we expect that commodities for which total demand is close to total supply will not have a big impact on finding the optimal route. When the total demand is close to the total supply of the product, there is not much potential profit that is missed by taking a different route. For example if the total demand is 105 and

the total supply is 100, we miss out on at most 5 times the profit value of this commodity at the last customer before the unloader. We expect there to be much more emphasis on minimization of travel cost for this subset and thus a high probability that the optimal route will be the shortest $s-t$ path. We expect it to be unlikely that this shortest $s-t$ path will visit the customers in such a way that these 2 commodities are always sold at the best available price. Therefore, this upper bound is expected to be somewhat tighter than taking the maximal profits and minimal cost.

Most profitable commodity We create a subset of a sole commodity with the highest potential profit. This means that the profit when selling the whole supply for this unit at the best available prices is the highest among all commodities. The reason for choosing this subset is that the highest potential profit is unlikely to come from the optimal route for this commodity. Therefore, expect this upper bound value to be significantly less than for the previous mentioned methods.

Two most profitable commodities Here we find the two commodities that provide the highest potential profits, following the same reasoning as choosing the most profitable commodity. We expect this method to provide us with the lowest upper bound value for all methods mentioned. We also expect this method to have the highest computational cost.

Since we have large amount of instances we limit the time we can look for an upper bound. We give CPLEX and the labeling algorithm a maximum of 1 minute per method to generate the optimal solution for the subsets. If an optimal value is not found by any of the methods we use the maximum profit minimum cost upper bound.

5 Data

In this subsection we discuss how we generate the data. This will be done for different instance sizes. We need to generate data for the demands and profits for all commodities in set P and data for the location of all customers in set N. We will create sets where $|N| = 10, 20, 30$ and $|P| = 2, 5, 10, 20$.

5.1 Demands and profits

The data that will be used will be randomly generated. The total capacity will always be $100 \cdot |P|$ and the starting supply for every product will always be 100. Demands for $|P| - 1$ commodities will be drawn from a truncated normal distribution with mean $\mu_p \cdot \frac{100}{|N|}$ where μ_p is uniformly distributed over $[1, 2]$. The standard deviation will be set to $\frac{50}{|N|}$ in order to diversify the demand. The total demand for these $|P|-1$ items must exceed the total starting capacity for this commodity, if this is not the case, the demand data is regenerated. The control commodity will be added by giving every customer a demand of $\left\lfloor \frac{100}{|N|} \right\rfloor$. As mentioned, the unloader has no specified demand. Because he needs to buy up the left over inventory we set his demand manually to the same amount as the total supply of this product.

For profits we randomly pick a mean θ_p from a uniform distribution between 0.5 and 3.5. We then generate profits from a truncated normal distribution around this mean with the standard deviation set to half of this mean. The profits at the unloader will be set to 0.001.

5.2 Customer locations

We will generate $|N|$ vertices in a Euclidean square with height 100 and width 100. The depot will always be centered (at (50, 50)). The x and y coordinates of the vertices will be generated form a uniform distribution between 0 and 100.

5.3 Instances

Our instances involve a set of customers and a set of items. We want to test our methods on a variety of instances. Therefore, we generate 5 different customer location sets per customer size and 10 commodity sets per commodity size. This means that we have 50 different instances for every combination of customer size and commodity size. We define a n-p instance as an instance involving n customers and p commodities. For example a 10-2 instance involves 10 customers and 2 commodities.

6 Results

In this section the results of the exact methods and the heuristics are presented. We test on a number of different instances involving different combinations of the number of customers and the number of products, in order to find the best solution method for that instance. We decide to divide all instances in to 4 different categories. Small-small instances are instances where the number of customers as well as the number of products is low. Small-large and large-small are instances where the number of customers is low (resp. high) and the number of products is high (resp. low). Large-large instances involve all instances with a high number of customers as well as products. We define a small number of customers as up to 15 customers and a large amount as more than 15 and a small number of products as 2 and a large amount as more than 2. An overview of the instance types can be found in Table 1.

Table 1: Different types of instances

Type of instance	**Number of customers**	**Number of products**
Small-small	≤ 15	2
Small-large	≤ 15	>2
Large-small	>15	2
Large-large	>15	>2

For every type of instance we want to compare the solution methods to see which method finds the best routes.

6.1 Small-small instances

We first examine 10-2 instances. These are the only instances for which an optimal value is found by the branch-and-bound method, the DP approach and the labeling algorithm. The mean and standard deviation (SD) of the running times for these methods are found in Table 2. From this table we can see that all methods are fast. Out of all three instances the label correcting algorithm is the fastest and most consistent algorithm for finding the optimal value. We can see that the DP approach also finds optimal routes in a fast and consistent manner. The running times for the branch-and-bound method are a lot more varied. They range from 0.049 seconds up to 15.428 seconds. This is to be expected since solving this problem with the branch-and-bound method involves some randomness when it comes to run time. CPLEX can find the right branch very quickly but in the worst case, all branches are evaluated which ups the running time considerably.

Table 2: running times in seconds for 10 customers and 2 commodities

Method	**Mean**	**SD**
branch-and-bound	2,05	4,51
dynamic programming	0,48	0,23
label correcting	0,072	0,013

When the number of customers is increased to 15, the branch-and-bound method and the label correcting algorithm are still able to quickly find the optimal route. The dynamic

programming approach no longer provides an optimal route within reasonable time, for the instances tested the average running time was around 40 minutes. For the label correcting algorithm, we found that increasing the number of customers causes an exponential increase in run time, which is to be expected. With n customers we have $(n-1)!$ different routes to consider. The consequence of increasing the number of customers to $n+1$ is thus that the amount of different routes to consider is multiplied by n. We ran the label correcting algorithm for instances involving 11-15 customers and 2 products. Our findings are presented in Table 3. Here we can see that the running time increases exponentially. We can also see that that the number of generated labels as well as the number of dominated labels more than doubles for every customer added.

Table 3: Data for the label correcting algorithm with 11-15 customers

Customers	**Run time**	**Labels generated**	**Labels dominated**
11	0,47	23040	17930
12	1,32	56320	45067
13	6,62	135168	110604
14	38,45	319488	266253
15	200,67	745472	229348

We are able to find the optimal value for all 15-2 instances by using branch-and-bound and the label correcting algorithm. When we compare the results for these two methods, which are presented in Table 4, we can see that the branch-and-bound method is much faster. The high standard deviation of the running time is due to 1 instance taking 71 seconds. Without this outlier the average would be 0.70 seconds (with a standard deviation of 0,81 seconds).

Table 4: Running times in seconds for 15 customers and 2 commodities

Method	**Mean**	**SD**
branch-and-bound	2,11	10,80
label correcting	200,67	13,87

Based on these findings, we can conclude that the increase in the number of customers, when the number of products is small, has less impact on the branch-and-bound method than on the label correcting algorithm and especially the dynamic programming approach.

Lastly we want to know how well heuristics H1 and H2 perform on the small customers, small products instances. The running times for 10-2 and 15-2 instances for H1 and H2 are given in Table 5

Table 5: Running times in seconds for H1 and H2 for 10 and 15 customers and 2 products

	H1		**H2**	
Customers	**Mean**	**SD**	**Mean**	**SD**
10	0,014	0,015	0,035	0,034
15	0,72	0,90	0,26	0,016

H1 and H2 were able to find the optimal route for all small instances. They both outperformed the exact methods. We can see that the increase in the number of customers leads

to a bigger increase in running time for H1 with respect to H2. Based on these results we expect H2 to outperform H1 when the number of customers increases.

Based on our findings, we can conclude that the best method for finding optimal routes for small-small instances depends on the number of customers. If the number of customers is 10, the label correcting algorithm is the best method since it is very consistent and fast. When the number of customers increases to 15 branch-and-bound becomes the best exact method since the increase in the number of customers has less effect on the running time of this method than on the label correcting algorithm and the DP approach, making it the fastest exact method for these specific instances. We also found that H1 and H2 are able to provide the optimal route for all small-small instances. Given that using these heuristics does not guarantee an optimal route it is better to use one of the before mentioned exact methods since the guarantee of optimality comes at a low computational cost for small-small instances. Next we want to test the effect of increasing the number of products on our methods.

6.2 Small-large instances

We first increase the number of products for all instances involving 10 customers. We find that for the branch-and-bound method increasing the number of products has a big impact on the running time. We found that for 10-5 instances the method could not find the optimal value for 5 out of the 50 instances within 5 minutes. The average running time was 67.6 seconds (standard deviation 90.2 seconds). Increasing the number of products does not seem to have a large impact on the DP approach and the label correcting algorithm. We know from section 6.1 however, that when we increase the number of customers to 15, the DP algorithm no longer gives us optimal routes within reasonable time. Therefore, we conclude that the label correcting algorithm is the best exact method to solve small-large instances. We first use this algorithm to solve all instances involving 10 customers. The data on all these instances are presented in Table 6, where we find the mean and standard deviation of the running time in seconds, the average number of dominated labels and the average number of generated labels.

Table 6: Data on all instances involving 10 customers solved by the label correcting algorithm

Commodities	**Mean**	**SD**	**Labels generated**	**Labels dominated**
2	0,072	0,013	9221.32	6999,56
5	0,067	0,0027	9198.88	6993,68
10	0,077	0,0029	9090.56	6974,66
20	0,10	0,0041	9088.14	6952,78

We can see that the label correcting algorithm runs very fast for all instances. Note that the number of generated labels as well as the number of dominated labels stay roughly the same when increasing the number of products. Increasing the number of products beyond 20 does not seem to have large impact on the running time either. We did some tests for 10-100 instances and found that the running time of the algorithm varied between 0,32 and 0,56 seconds. We also found that H1 and H2 where again able to find optimal values for all instances tested and outperformed the label correcting algorithm. Since the running times

are so low, the guarantee of optimality again comes at a low computational cost and for these instances, using the label correcting algorithm (or DP for that matter) is preferred.

The benefit of using the heuristics becomes clear when examining all instances involving 15 customers. We know that the running time for the label correcting algorithm increases exponentially with an increase in the number of customers. The increase in the number of customers does not seem to have such a large impact on the running time for the heuristics. They are both able to find the optimal values for all instances tested. It seems that H2 is somewhat faster than H1, as can be seen in Table 7. This reinforces our earlier mentioned believes that H2 will outperform H1 when the number of customers increases.

Table 7: Running times in seconds for the LCA, H1 and H2 for 15 customers

	LCA		**H1**		**H2**	
Commodities	**Mean**	**SD**	**Mean**	**SD**	**Mean**	**SD**
2	200,67	3,76	0,72	0,90	0,26	0,16
5	219,04	13,87	1,19	1,68	0,42	0,36
10	227,59	17,06	2,26	2,81	0,66	0,61
20	242,48	20,45	3,82	6,17	1,74	2,02

The guarantee of optimality now comes at a higher computational cost. If the salesman has enough time, again using the label correcting algorithm is preferred since the running time is not extremely high. (We also tested a few 15-100 instances and found the average running time to be around 600 seconds, optimal values where found by H1 and H2 within a minute). If the salesman wants to know a good route in a few seconds, using H1 or H2 will be very likely to provide the optimal route.

We have already seen the impact from increasing the number of customers from 10 to 15, next we want increase the number of customers even further and see what impact this has on our methods. We first want to do this with instances involving a small number of products.

6.3 Large-small instances

We evaluate all instances involving 20 or more customers and 2 products. We already know that the DP approach will not give us optimal values within reasonable time. We also know that the run time of the label correcting algorithm increases exponentially. This algorithm did not provide an optimal route within an hour for instances involving 20 customers. We also know from the small-small instances that increasing the number of customers from 10 to 15 did not have a big impact on the branch-and-bound method. Based on all these findings we conclude the branch-and-bound method is the only method that could be capable of finding optimal routes within reasonable time.

We thus test the effect of increasing the number of customers on the branch-and bound method. We found that CPLEX was able to find the majority of optimal values within 5 minutes. In Table 8 we present the number op optimum values found by branch-and-bound, H1 and H2 for the different amounts of customers in large-small instances. Note that H1 and H2 can only find optimum values if branch-and-bound provides us an optimal value.

Table 8: Number of optimal values found for 20, 25 and 30 customers

Customers	Instances	BB	H1	H2
20	50	39	37	39
25	50	37	1	35
30	50	40	0	20

We expect that most of the unsolved instances can be solved within a reasonable time since the gap of the lower bound and upper bound of the optimal value found by CPLEX was less then 5% for most instances. We also found that for instances involving 25 customers or more that H1 could not consistently provide optimal values within a minute. The same applied to H2 for instances involving more than 30 customers. We tested these instances with a longer time frame to check whether or not H1 and H2 where able to find optimal values. The optimal value was found for most instances tested, but it could take up to 10 minutes. If an optimal route was not found by H1 and H2, they returned routes which value was within 4% of the optimal value for all instances for which an optimal value is known. H3 performed at least as good for every instance and the values found were at most 2% from the optimum.

Large-small instances are the first type of instances for which our methods can not consistently provide optimal routes. The DP approach and the label correcting algorithm can not provide optimal routes within reasonable time. The branch-and-bound method is likely to provide the optimal result within reasonable time, but it can not provide an optimal route for all instances tested. We find that when the number of customers grows, H1 and H2 are less consistent in providing (near) optimal results. This is to be expected since an increase in customers means an exponential increase in the number of possible routes. This is likely to lead to a high number of local optima. Since H1 and H2 start with a random route, the probability that the local optimum they end up in is the global optimum, would be less if the number of local optima is higher. Still, when given more time H1, H2 and especially H3 provide very good results.

6.4 Large-large instances

When it comes to the larger instances involving more than 15 customers and at least 5 products, we are no longer able to find optimal routes within reasonable time with the exception of 11 20-5 instances, found by CPLEX (for which H2 was able to find the optimum value within a minute for all instances). Thus for large-large instances we need our heuristics to provide good routes. In this section we first want to see how close we are to finding the optimal value for the different large-large instances by examining the difference between the upper bound and lower bound on the optimum value provided by CPLEX. Then we want to compare H3 to the upper bounds we have found as described in section 4.3 to see how H3 performs.

6.4.1 Upper and lower bounds on the optimal value

We first check the average gap between the upper and lower bound that is found by CPLEX. In Table 9, the average gap over all 50 instances of all different combinations of number of

customers and number of products are presented. From the table we can see that the gap is usually around 15-25%. Due to the nature of branch-and-bound this does not mean that we can be certain that we are close to finding the optimal value. Since closing this gap would in the worst case still require all unvisited branches to be searched completely.

Table 9: Gap in percentage between the upper and lower bound of the optimal value

Customers/Commodities	5	10	20
20	15,6	17,7	16,0
25	14,2	14,8	23,9
30	16,0	17,9	16,3

We suspect that for the FSP it is easy for the branch-and-bound method to find the optimal value, but it is very hard to prove that this value is optimal. The reason for our suspicion is the fact that, for the smaller instances, CPLEX is able to very quickly find the optimal value and return that as a lower bound. The majority of the running time comes from pushing down the upper bound to where it meets the lower bound and it is proven that this lower bound is indeed the optimal value. In large-large instances we see that the lower bound of the optimal value found by CPLEX is often the same value given by H3. This is especially the case in all instances involving 20 and 25 customers, for 30 customers the values start to differ significantly. Since the heuristics provided very good results for the smaller instances we have no reason to assume that they do not provide reasonable results for the large-large instances.

6.4.2 Finding the best upper bound and comparing this upper bound to H3

We first find the upper bounds as described in section 4.3 and compare this bound to the value we got from H3. The method in which the upper bound was found, the number of times this method occurs across all large-large instances, the average and the standard deviation of the gap between the found upper bound and H3 are presented in Table 10. Here method 1 refers to the two most profitable commodities method, 2 to the most profitable commodity method, 3 to the two least demanded commodities method and 4 to the maximum profit minimum cost method. We can see that the gap between the upper bound that was found and the value found by H3 is very dependent on the method by which the upper bound was found. If the upper bound is tighter (with method 1 being the tightest) the gap becomes smaller. Therefore, we expect that if the upper bound for all instances was found by means of method 1, the average gap between H3 and the upper bound would be much smaller. This indicates that H3 provides very good routes that we suspect are close to the optimal value.

Table 10: Percentual difference between found upper bound and H3 per method

Method	**Occurrences**	**Mean**	**SD**
1	42	4,7	0,3
2	98	6,4	2,0
3	133	6,8	3,9
4	177	8,8	5,8

Next we compare the gaps between the found upper bound and H3 per customer-commodity combination. The upper bounds used here are the minimum over the upper bounds obtained by the methods described in 4.3. The average difference and standard deviation (in brackets) of all 50 instances per combination are presented in Table 11.

Table 11: Percentual difference between the upper bound and H3 per customer/commodity combination

Customers/Commodities	5	10	20
20	7,5 (6,5)	7,7 (3,3)	9,2 (3,4)
25	6,5 (3,4)	6,4 (1,4)	9,2 (2,6)
30	6,1 (2,7)	5,4 (1,4)	5,5 (0,8)

We can see that the average difference between the upper bound and best value returned by H3 is between 5.5 and 9.2 %. The standard deviation of 20-5 is high which is due to a substantial amount of optimal values found and some instances having a high difference. We can not conclude that the difference is going up when the number of products increases, since we see this value going up for 20 and 25 customers, but down for 30 customers.

We do see the average values go down when the number of customers is increasing. This is not due to instances involving a higher number of customers having tighter upper bounds. On the contrary, almost all upper bounds for instances involving 30 customers are found by method 4. A reason for this decrease could be that when the number of customers increases, the demand per customer decreases due to the nature in which the data was generated. It thus becomes less likely to loose out on a big profit when the supply for a product is depleted.

This is best illustrated by an example. Consider an instance with 5 customers. In this scenario a customer usually demands around 25 units per commodity. In the optimal route we arrive at a certain customer c while the stock of product p is depleted. This is unfortunate because c was prepared to pay 5 per unit of p, which is the most of all customers. Thus, in this route we miss out on 125 potential profit generated from customer c.

Now let that same customer c be part of a route involving 30 customers. The average demands will be much lower in a route involving 30 customers. The demand of c for product p is only 5 in this instance. So when we again arrive at c with a depleted stock, we only miss out on 25 potential profit.

However, since c was the highest paying customer for product p, these profits will be included in the calculation of the upper bounds (at least in method 4). Thus, missing out on these profits will most likely lead to a larger gap between the upper bound and the optimal value for instances involving a smaller number of customers.

When comparing the upper bounds to H3 we can conclude that this heuristic provides us with very good routes. We expect that the gap between H3 and the optimal value will be very small for most instances. Allowing more time to calculate tighter upper bounds and running the heuristic for a longer amount of time could provide us with an even more clear insight on how close H3 comes to the optimal value.

6.5 Comparison with naive heuristics

Since we cannot conclude that H3 consistently provides (near) optimal solutions for larger instances, we want to see how well the heuristic performs compared to some naive heuristics that may be selected by a real-life salesman based on the data he possesses. We want to compare H3 to a greedy profit heuristic and a greedy distance heuristic. First we examine the difference between the greedy profit heuristic (GP) and H3. The average and standard deviation (in brackets) for the percentual difference between H3 and GP are presented in Table 12

Table 12: Percentual difference between H3 and GP

Customers/Commodities	2	5	10	20
10	239,2 (272,9)	26,5 (12,5)	13,8 (4,8)	5,0 (2,1)
15	771,8 (740,13)	67,3 (24,9)	25,2 (6,3)	10,1 (2,1)
20	982,6 (1091,9)	96,5 (42,5)	38,3 (7,6)	16,4 (4,1)
25	1107,9 (655,9)	177,7 (94,8)	52,3 (10,6)	26,0 (5,1)
30	1316,2 (1447,0)	273,8 (126,7)	72,0 (12,4)	30,5 (4,0)

We can see from the table that the GP does perform very badly, especially for 2 commodities. It starts to perform somewhat better when the number of commodities rise. We suspect the following reason for this improvement: when dealing with a small number of commodities, it is very likely that one customer is very profitable for one commodity but very bad for the majority of the other commodities. For example let there be 5 commodities. If a customer demands 20 for every commodity and pays 4 for commodity 1 and 1 for commodities 2-5 for a total of 120, he will be selected before a customer demanding 5 and paying 20 for every commodity for a total of 100. This could lead to huge amounts of missed profits. Since there are more products the likelihood of a very bad customer being visited before a good customer becomes smaller since there is more spread in the profits. Although the performance of GP becomes better with the increase in the number of commodities we observe that even for 20 commodities H3 performs much better than GP.

We also compare the value of H3 to a greedy distance heuristic (GD). The average and standard deviation (in brackets) for the percentual difference between H3 and GP are presented in Table 13

Table 13: Percentual difference between H3 and GD

Customers/Commodities	2	5	10	20
10	42,7 (47,4)	11,7 (8,9)	6,4 (4,0)	4,4 (2,8)
15	77,2 (76,2)	9,1 (6,2)	5,1 (3,6)	3,2 (2,0)
20	93,5 (77,9)	18,3 (8,5)	8,4 (2,9)	4,2 (1,5)
25	84,6 (61,5)	14,6 (8,3)	4,8 (2,0)	3,6 (1,7)
30	131,5 (192,3)	21,9 (15,1)	6,4 (3,0)	3,1 (1,5)

GD clearly performs better than GP. We see that performance again increases with the number of commodities. For GD we suspect that this improvement in performance is due

to the lower risk of visiting a low paying customer before a high paying customers. This route is chosen solely based on traveling cost. When the number of commodities is low, there is a higher probability that a customer that pays very low prices for all commodities is visited before customers who pay higher prices for these commodities. If there are more commodities, the probability of a customer paying low prices for most or even all commodities will decrease significantly. If a customers has demands for 20 different commodities, there is a high probability that a relatively low revenue gathered from this customer for a certain product will be offset by a relatively high revenue gathered from a different product.

We can see that especially for the larger number of commodities GD comes very close to H3. This would mean that if H3 provides very good results that GD is also capable of providing decent results for the instances tested. The better option would still be H3 since 1 minute of computation time provides us with on average at least a 3% increase in profits. Meaning that at minimal computing cost a lot of additional profit can be made. Note that H3 gave a better route than both heuristics for every instance. We conclude that H3 could be a very good heuristic that can easily be implemented by a real-life salesman.

7 Conclusion and discussion

In this section we sum up our findings and provide points of interest for future research.

7.1 Conclusion

In this thesis we introduced the flower seller problem, which can be seen as an extension of the TSP. We gave a mathematical formulation of the problem which was used to solve instances using the commercial solver CPLEX. This method proved to be useful to solve most instances involving 2 commodities. We applied a dynamic programming approach to the problem which was able to quickly give the optimal route for instances involving 10 customers. However, when the number of customers went beyond 10, this approach no longer was viable. We also introduced a label correcting algorithm based on Feillet et al. (2004), with a few adjustments to the dominance rules. This algorithm provided us with optimal results for all instances with up to 15 customers within 5 minutes.

Since larger instances would take much longer to solve to optimality (if even possible) we introduced heuristics to provide us with good routes. Heuristic H1 based on swapping similar nodes in the route provides fast optimal results for up to 15 customers. H2, which is based on the algorithm presented in Kanellakis and Papadimitriou (1980), also provides fast optimal results for up to 15 customers and started to outperform H1 for larger instances. The combination of H1 and H2, named H3 works even better than H2 and is used to get good routes in the larger instances. After computing the upper bounds for these instances we conclude that we assume it to be likely that H3 gives (near) optimal values for these instances. When comparing the results to naive heuristics that may be used by a real-life salesman we conclude that H3 always performs better (for instances concerning less commodities considerably) than these naive heuristics at a very low computational cost. We conclude that H3 could be a good heuristic to find profitable routes for large instances.

7.2 Further research

When considering the FSP in a real-life setting, one could approach the problem of the salesman from different angles. In this research we focused on optimizing routes when demands and profits are known. Different approaches that could be interesting is optimizing the inventory that the salesman takes with him. The optimization of the inventory could also be combined with finding the best route in a two step approach, for example one could search for the most profitable route with a constraint on the total amount of products that can be loaded in to the truck instead of constraints on each commodity. Another interesting extension would be stochastic demands or profits. In real-life the demand and profits will not be known beforehand but can be estimated. It would be interesting to find methods to optimize routes when a set of different demands and profits is known for every customer.

References

Altinel, I. K., Aras, N., & Oommen, B. J. (2000). Fast, efficient and accurate solutions to the hamiltonian path problem using neural approaches. *Computers & Operations Research*, *27*(5), 461–494.

Aneja, Y. P., Aggarwal, V., & Nair, K. P. (1983). Shortest chain subject to side constraints. *Networks*, *13*(2), 295–302.

Awerbuch, B., Azar, Y., Blum, A., & Vempala, S. (1998). New approximation guarantees for minimum-weight k-trees and prize-collecting salesmen. *SIAM Journal on Computing*, *28*(1), 254–262.

Balas, E. (1989). The prize collecting traveling salesman problem. *Networks*, *19*(6), 621–636.

Beasley, J. E., & Christofides, N. (1989). An algorithm for the resource constrained shortest path problem. *Networks*, *19*(4), 379–394.

Bellmore, M., & Nemhauser, G. L. (1968). The traveling salesman problem: A survey. *Operations Research*, *16*(3), 538–558.

Chao, I.-M., Golden, B. L., & Wasil, E. A. (1996). The team orienteering problem. *European Journal of Operational Research*, *88*(3), 464–474.

Dantzig, G., Fulkerson, R., & Johnson, S. (1954). Solution of a large-scale traveling-salesman problem. *Journal of the Operations Research Society of America*, *2*(4), 393–410.

Dell'Amico, M., Maffioli, F., & Värbrand, P. (1995). On prize-collecting tours and the asymmetric travelling salesman problem. *International Transactions in Operational Research*, *2*(3), 297–308.

Desrochers, M. (1988). *An algorithm for the shortest path problem with resource constraints*. École des hautes études commerciales, Groupe d'études et de recherche en analyse des décisions.

Dror, M. (1994). Note on the complexity of the shortest path models for column generation in vrptw. *Operations Research*, *42*(5), 977–978.

Feillet, D., Dejax, P., & Gendreau, M. (2005). Traveling salesman problems with profits. *Transportation Science*, *39*(2), 188–205.

Feillet, D., Dejax, P., Gendreau, M., & Gueguen, C. (2004). An exact algorithm for the elementary shortest path problem with resource constraints: Application to some vehicle routing problems. *Networks: An International Journal*, *44*(3), 216–229.

Gendreau, M., Laporte, G., & Semet, F. (1998). A tabu search heuristic for the undirected selective travelling salesman problem. *European Journal of Operational Research*, *106*(2-3), 539–545.

Held, M., & Karp, R. M. (1962). A dynamic programming approach to sequencing problems. *Journal of the Society for Industrial and Applied mathematics*, *10*(1), 196–210.

Helsgaun, K. (2000). An effective implementation of the lin–kernighan traveling salesman heuristic. *European Journal of Operational Research*, *126*(1), 106–130.

Kanellakis, P.-C., & Papadimitriou, C. H. (1980). Local search for the asymmetric traveling salesman problem. *Operations Research*, *28*(5), 1086–1099.

Kataoka, S., & Morito, S. (1988). An algorithm for single constraint maximum collection problem. *Journal of the Operations Research Society of Japan*, *31*(4), 515–531.

Laporte, G. (1992). The traveling salesman problem: An overview of exact and approximate algorithms. *European Journal of Operational Research*, *59*(2), 231–247.

Laporte, G., & Martello, S. (1990). The selective travelling salesman problem. *Discrete Applied Mathematics*, *26*(2-3), 193–207.

Lin, S., & Kernighan, B. W. (1973). An effective heuristic algorithm for the traveling-salesman problem. *Operations Research*, *21*(2), 498–516.

Madkour, A., Aref, W. G., Rehman, F. U., Rahman, M. A., & Basalamah, S. (2017). A survey of shortest-path algorithms. *arXiv preprint arXiv:1705.02044*.

McGill, J. I., & Van Ryzin, G. J. (1999). Revenue management: Research overview and prospects. *Transportation Science*, *33*(2), 233–256.

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

Pugliese, L. D. P., & Guerriero, F. (2013). A survey of resource constrained shortest path problems: Exact solution approaches. *Networks*, *62*(3), 183–200.

Ramesh, R., & Brown, K. M. (1991). An efficient four-phase heuristic for the generalized orienteering problem. *Computers & Operations Research*, *18*(2), 151–165.

Righini, G., & Salani, M. (2008). New dynamic programming algorithms for the resource constrained elementary shortest path problem. *Networks: An International Journal*, *51*(3), 155–170.

Rosenkrantz, D. J., Stearns, R. E., & Lewis, P. M., II. (1977). An analysis of several heuristics for the traveling salesman problem. *SIAM Journal on Computing*, *6*(3), 563–581.

Tsiligirides, T. (1984). Heuristic methods applied to orienteering. *Journal of the Operational Research Society*, *35*(9), 797–809.

Wren, A., & Holliday, A. (1972). Computer scheduling of vehicles from one or more depots to a number of delivery points. *Journal of the Operational Research Society*, *23*(3), 333–344.