DIPLOMARBEIT

Titel der Diplomarbeit

“The Traveling Repairman Problem
- A Review”

Verfasserin

Miriam Lechmann

Angestrebter akademischer Grad

Magistra der Sozial- und

Wirtschaftswissenschaften

(Mag. rer. soc. oec.)

Wien, im November 2009

Studienkennzahl lt. Studienblatt:	A 157
Studienrichtung lt. Studienblatt:	Internationale Betriebswirtschaftslehre
Betreuer:	O.Univ.Prof.Dipl.-Ing. Dr. Richard F. Hartl

I. TABLE OF CONTENTS

II. List of Figures

III. List of Tables

1 Introduction

When a salesman starts to plan his tour he has to decide in which order he wants to visit his customers. This he has to do in the most efficient way possible to minimize the time spent on traveling and the costs which are associated with this. Additionally he will be able to visit more customers in the given time limit if he is able to find the optimal tour and this will increase his profits, too.

This problem is called the Traveling Salesman Problem (TSP) and is one of the most frequently researched in the area of operations research because it has many applications not only in the above described way and is typical of other problems in combinatorial optimization. There exist a lot of variations of it depending on the variables that are included. One of them is the so called Traveling Repairman Problem (TRP).

This diploma thesis wants to give a review about the characteristics and the different model approaches of the TRP and its variations. In Chapter 2 a short overview of the TSP and the TRP is given, and in Chapter 3 there is a detailed description of some solving methods for the “classical” TRP. In Chapter 4 some variations of the TRP with a single repairman are examined while Chapter 5 describes variants of the TRP with several repairmen. This is followed by a summary of the possible applications for the TRP with the more detailed examination of two cases. In Chapter 7 some basis and solving methods for the TRP are outlined.

2 Problem Description and Definitions

As described above the Traveling Repairman Problem (TRP) is a special form of the Traveling Salesman Problem (TSP). So to examine the TRP it is necessary to investigate the characteristics of the TSP first.

2.1 The Traveling Salesman Problem (TSP)

Every day a traveling salesman faces the challenge of planning his tour. He has to visit a given number of customers exactly once starting from his home city or a depot and to return at the end of the day. Naturally he tries to select the order of the customer visits in a way which allows him to spend as little time as possible on traveling. In other words he wants to find the tour which **minimizes the total of the distances to save time and costs**.

The problem can be defined on a graph $G=(V,E)$ with a set of n vertices $V=\{v_1,\ v_2,\ \dots,\ v_n\}$ and an arc set $E \subseteq V \times V$ (see Figure 1). The vertices correspond to the customers or the cities the traveling salesman has to visit during his tour while the arcs represent the streets he may use to get to them. It is assumed that the distances between each pair of customers are known in advance, so for each arc $(v_i,\ v_j)$ there are known costs c_{ij}. If there is no connection between customer i and customer j, then c_{ij} will be set to infinite.

The underlying graph of the TSP can take a lot of different forms depending on the problem formulation or on the characteristics of the problem one tries to solve. It could be a line, a directed or undirected graph, the metric space and so on.

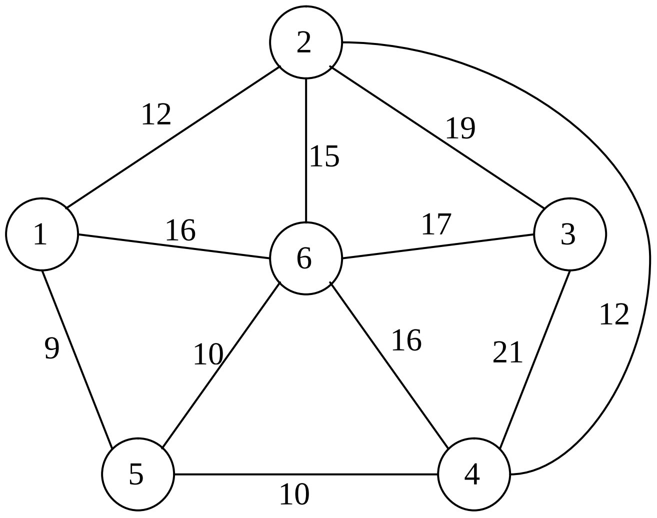

Figure 1: Example of a graph[1]

The mathematical formulation is as follows:[2]

$$\sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij} x_{ij} \rightarrow \min \tag{2.1}$$

As stated above the objective is to minimize the total of the distances.

$$x_{ij} \in \{0,1\} \tag{2.2}$$

For this purpose the binary variable x_{ij} is introduced which determines whether the vertex j is visited right after the vertex i ($x_{ij} = 1$) or not ($x_{ij} = 0$).

$$\sum_{j=1}^{n} x_{ij} = 1 \qquad for\ i = 1,\ \ldots,\ n \tag{2.3}$$

To guarantee that each vertex is visited only once two further equations have to be introduced. First every customer must have exactly one successor.

[1] Sarubbi et al., 2007, p. 1

[2] Lawler et al., 1985, pp. 25-26

$$\sum_{i=1}^{n} x_{ij} = 1 \qquad for\ j = 1,\ \ldots,\ n \tag{2.4}$$

Then it has to be ensured that every customer also has a predecessor.

Additionally there has to be a constraint which forbids any subtours. It could be for example

$$\sum_{i\in S}\sum_{j\in S} x_{ij} \le |S| - 1 \tag{2.5}$$

for every proper, nonempty subset $S \subseteq N = \{1,\ \ldots,\ n\}$ with $|S|$ denoting the cardinality of S.

The solution of the example above is depicted in Figure 2. The optimal TSP-tour has a length of $9+10+17+21+12+12=81$.

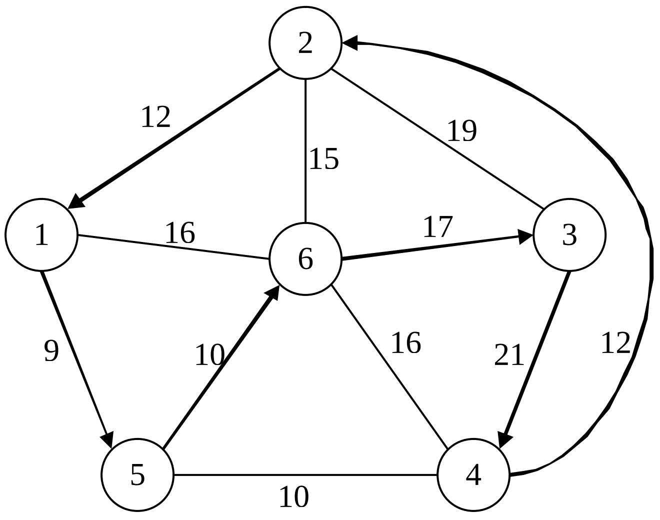

Figure 2: TSP solution[3]

Although the TSP seems to be simple it is rather difficult to solve optimally. There exist $(n-1)!$ possible tours, so even if the number of costumers n is relatively small there is a huge number of possible solutions (see Table 1).

[3] Sarubbi et al., 2007, p. 1

n	$(n-1)!$
3	2
4	6
5	24
6	120
7	720
8	5.040
9	40.320
10	362.880

Table 1: Number of possible tours with a given number of customers

Therefore it seems to be impossible to solve the TSP optimally through complete enumeration during an acceptable amount of time if the value of n is rather high. It is one of the most prominent problems that is NP-hard for the general metric.

2.2 The Traveling Repairman Problem (TRP)

The TRP is a variant of the TSP with the same task but the objective is a different one. It looks at the same problem from another point of view by turning its attention more to the satisfaction of the customers. This could be necessary because the order of the customer is urgent and therefore needs to be completed as soon as possible. So the objective is not to minimize the traveling time of the repairman but to minimize the sum of the latencies of the customers. For this reason the TRP is often also referred to as the Minimum Latency Problem in the literature. There also exist other names for it, for example the “Delivery Problem”, the “Deliveryman Problem” or the “TSP with Cumulative Costs”.

2.2.1 Definition

The TRP can be mathematically formulated as follows.[4]

$$\sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij} x_{ij} \rightarrow \min \qquad (2.6)$$

The objective function looks the same as above but the variables are defined differently and therefore it is subject to other constraints.

$$y_{ij} \in \{0,1\} \qquad (2.7)$$

$$\sum_{j=1}^{n} y_{ij} = 1 \qquad for\ i = 1,\ \ldots,\ n \qquad (2.8)$$

$$\sum_{i=1}^{n} y_{ij} = 1 \qquad for\ j = 1,\ \ldots,\ n \qquad (2.9)$$

These three constraints introduce the necessary binary variable which should indicate whether arc $(v_i,\ v_j)$ is part of the tour and make sure that each vertex will be visited.

[4] Fischetti et al., 1992, pp. 1055-1056

$$\sum_{i=2}^{n} x_{i1} = 1 \tag{2.10}$$

Equation (2.10) makes sure that the starting point v_1 has exactly one predecessor, so that the tour ends at the origin.

$$\sum_{i=1}^{n} x_{ik} - \sum_{j=1}^{n} x_{kj} = \begin{cases} 1-n & \text{for } k = 1 \\ 1 & \text{for } k = 2, \ \dots, \ n \end{cases} \tag{2.11}$$

Contraint (2.11) ensures that there will be no subtour which is disconnected of the starting point v_1 and gives x_{ij} the value $n-k+1$ if arc $(v_i, \ v_j)$ appears in position k on the tour together with (2.10).

$$x_{ij} \geq 0 \tag{2.12}$$

This is the necessary non-negativity constraint for x_{ij}.

$$x_{ij} \leq r_{ij} y_{ij} \tag{2.13}$$

with

$$r_{ij} = \begin{cases} 1 & \text{if } \ j = 1 \\ n & \text{if } \ i = 1 \\ n-1 & \text{otherwise} \end{cases} \tag{2.14}$$

Constraint (2.13) makes sure that x_{ij} only takes a positive value if y_{ij} is positive and the variable r_{ij} represents an upper bound on x_{ij}.

The difference of the TRP-model to the one for the TSP is that the distances from one vertex to another are not simply added to get a final solution but that for example the distance from the starting point to the first vertex must be added $n-1$ times as it will influence the latencies of all the other customers. Therefore the variable x_{ij} will take either the value 0 if the corresponding edge is not part of the tour or the value $n-k+1$ if it lies on position k of the tour.

The solution of the TRP with the same underlying graph as in Chapter 2.1 is $9+19+31+50+67+83=259$ (see Figure 3). The optimal TRP-tour has a length of 83 which is slightly longer than the TSP-tour, but the sum of the customers' latencies are smaller than the one of the TSP ($9+19+36+57+69+81=271$).

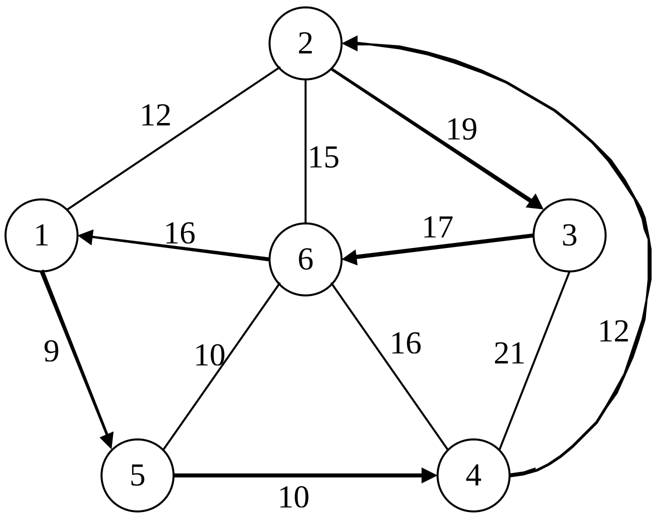

Figure 3: TRP solution[5]

Naturally the TRP is NP-hard for the general metric as well because it is a variant of the TSP, but in some aspects it is also very different from the TSP. If there is only a small change in the structure of a metric space, this can lead to highly non-local changes in the structure of the TRP, which would not be the case for the TSP. As we will discuss later in Chapter 4.1 this has a huge effect for example for the Line-TRP.

[5] Sarubbi et al., 2007, p. 2

3 A Solution Method for the “Classical” TRP

For a better understanding we now examine a few approaches for solving the “classical” TRP at more detail. “Classical” means that the underlying graph is a metric space and that there are no more constraints than described above in Chapter 2.2.

In 1993, Bianco et al. developed two exact algorithms using lower bounds generated by a Lagrangean relaxation of the problem. The first one is a branch and bound approach while the second uses dynamic programming to reduce the dimensionality of the state space graph. Additionally they presented a heuristic procedure that is also able to calculate the distance from the optimal solution.[6]

3.1 Notation and Definitions

First we introduce the variable l_k indicating the distance from the origin to v_{i_k} (i.e. the latency), the vertex occupying the position k in the tour $H = \{v_{i_1},\ v_{i_2},\ \ldots,\ v_{i_n},\ v_{i_{n+1}}\}$. As the traveling repairman has to return to the starting point, we have to add the vertex $v_{i_{n+1}}$ to the tour which represents the starting point v_{i_1} at the same time. The costs $c_{i_k i_{k+1}}$ represent the distance between the two vertices on positions k and $k+1$. Therefore l_k is given by

$$l_1 = 0, \quad l_2 = c_{i_1 i_2}, \quad l_3 = c_{i_1 i_2} + c_{i_2 i_3}, \quad \ldots, \quad l_k = c_{i_1 i_2} + c_{i_2 i_3} + \ldots + c_{i_{k-1} i_k}, \quad \ldots,$$
$$l_{n+1} = c_{i_1 i_2} + c_{i_2 i_3} + \ldots + c_{i_{k-1} i_k} + \ldots + c_{i_n i_{n+1}}$$

and the cost $z(H)$ of the tour H is given by

$$z(H) = \sum_{k-1}^{n+1} l_k = \sum_{k=1}^{n} (n-k+1)\ c_{i_k i_{k+1}}\ .$$

[6] Bianco et al., 1993, p. 81-91

Now we divide the tour at the position s into two paths F and B (see Figure 4) so that $H = F + B$, where the “forward path” is given by

$$F = \{v_{i_1},\ v_{i_2},\ \ldots,\ v_{i_{s-1}},\ v_{i_s}\}$$

and the “backward path” is given by

$$B = \{v_{i_s},\ v_{i_{s+1}},\ \ldots,\ v_{i_n},\ v_{i_{n+1}}\} = \{v_{j_1},\ v_{j_2},\ \ldots,\ v_{j_B},\ v_{j_{B+1}}\}$$

with

$$v_{j_1} = v_{i_s},\ \ \ldots\ \ ,\ v_{j_{B+1}} = v_{i_{n+1}}\ .$$

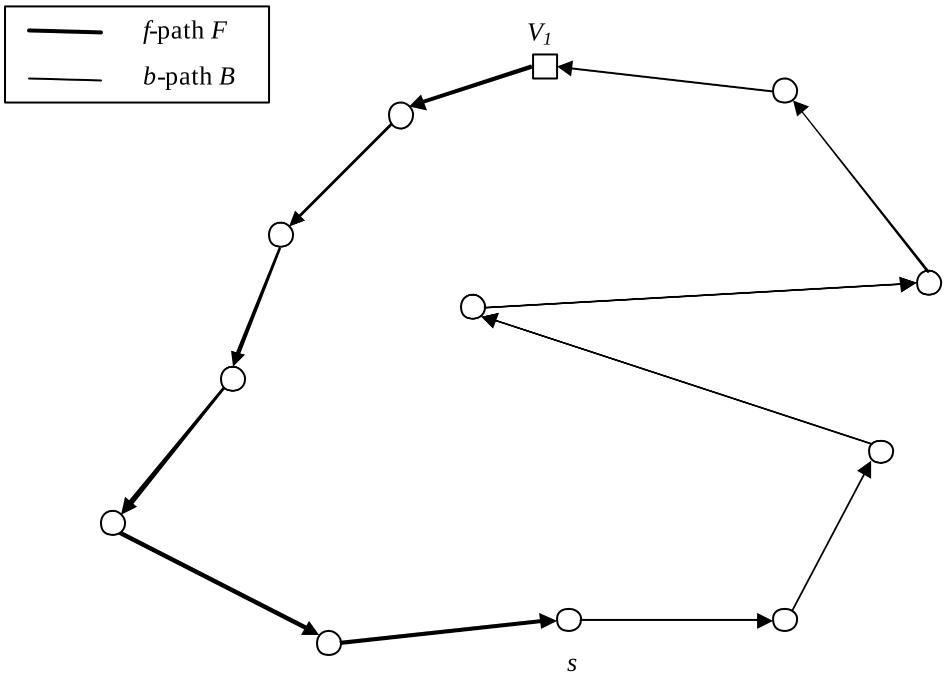

Figure 4: Example of a tour divided into f -path and b -path

3.2 Problem Formulation

To describe the problem we introduce the binary variable x_{ij}^k which takes the value 1 if the arc $(v_i,\ v_j)$ is in position k on the tour and 0 otherwise. If it occupies this position, the costs of adding it to the tour are indicated by $p(k)c_{ij}$ with $p(k)=(n-k+1)$.

The mathematical formulation is[7]

$$z=\sum_{k=1}^{n} p(k)\sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij}^k \rightarrow \min \tag{3.1}$$

The objective function minimizes the sum of the customer's waiting times.

$$\sum_{j=1}^{n} x_{ij}^k - \sum_{l=1}^{n} x_{li}^{k-1} = 0 \qquad \mathit{for}\ \ i,\ k=2,\ ...,\ n \tag{3.2}$$

This constraint makes sure that the tour will not be interrupted, so that for every customer there exists a successor and a predecessor.

$$\sum_{j=1}^{n} x_{1j}^1 = 1 \tag{3.3}$$

$$\sum_{i=1}^{n} x_{i1}^n = 1 \tag{3.4}$$

These equations state that the tour has to start and end at the starting point v_1.

$$\sum_{k=1}^{n}\sum_{i=1}^{n} x_{ij}^k = 1 \quad \mathit{for}\ j=1,\ ...,\ n \tag{3.5}$$

$$\sum_{k=1}^{n}\sum_{j=1}^{n} x_{ij}^k = 1 \quad \mathit{for}\ \ i=1,\ ...,\ n \tag{3.6}$$

$$\sum_{i=1}^{n}\sum_{j=1}^{n} x_{ij}^k = 1 \quad \mathit{for}\ \ k=1,\ ...,\ n \tag{3.7}$$

[7] Bianco et al., 1993, p. 82

These constraints make sure that each vertex will be visited exactly once and that on each position k there will be exactly one edge. As the tour starts at position $k=1$, then crosses all the other edges in the order of their positions until it reaches $k=n$, equation (3.7) prevents any subtours.

$$x_{ij}^{k} \in \{0,1\} \tag{3.8}$$

Finally we also have to determine that x_{ij}^{k} can only take the values 0 or 1.

This model varies from the one of Chapter 2.2 only at first sight. In fact it just defines the variables differently which makes it necessary to formulate the constraints in another way.

3.3 Branch and Bound Algorithm

3.3.1 Lower Bounds

As this model is too complex to be solved for a problem with numerous vertices, Bianco et al. derived lower bounds with the help of a Lagrangean relaxation.[8] First equations (3.5) and (3.6) are dualized, so that the relaxed problem is

$$RP \begin{cases} \sum_{i,j,k=1}^{n} \left[p(k)c_{ij} + l_i + l_j\right] x_{ij}^{k} - 2\sum_{i=1}^{n} l_i \rightarrow \min \\ \textit{subject to constraints } (3.2),\ (3.3),\ (3.4),\ (3.7),\ (3.8) \end{cases}$$

Obviously every feasible solution of RP will be a path visiting all n vertices, that starts and ends at v_1. We solve this problem by dynamic programming. First we introduce the variable $h\,(k,v_i)$ as the cost of the shortest b-path with the starting point v_i and the ending point v_1 which visits exactly k vertices. It can be computed recursively by

$$h\,(k,v_i) = \min_{v_j \in V - \{v_i\}} \left\{h\,(k-1,v_j) + c_{ij}^{k}\right\} \qquad \forall\ k = 1,\ ...,\ n$$

with $c_{ij}^{k} = kc_{ij} + l_i + l_j$. We initialize the recursion by

$$h\,(0,v_i) = \begin{cases} 0 & \textit{if } v_i = v_1 \\ \infty & \textit{for all } v_i \neq v_1 \end{cases}$$

Now we see that the lower bound LB, which corresponds to the optimal solution we derived from RP, is

$$LB = h\,(n,v_1) - 2\sum_{i=1}^{n} l_i\ .$$

[8] Bianco et al., 1993, pp. 83-84

Because this solution may include loops when constraints (3.5) and (3.6) get violated, we define the following penalties l_i for each vertex v_i.

$$l_i = l_i + d \frac{(z_{UB} - LB)\ (d_i - 2)}{\left[\sum_{j=1}^{n} (d_j - 2)^2\right]^{1/2}}$$

where d is a constant and z_{UB} is an upper bound for the original problem (see Chapter 3.3.2). The variable d_i represents the degree of a vertex i, i.e. the number of edges connected to it that are used in the tour. This leads to a penalty if a vertex is visited more than once ($d_i > 2$).

When starting the algorithm we define $l_i = a_i + b_j$ with $a_i = \min_j \{c_{ij}\}$ and $b_j = \min_i \{c_{ij} - a_i\}$. Naturally we have to update l_i at each iteration.

3.3.2 Upper Bound

Now we have to derive the upper bound z_{UB} by using the "nearest neighbor" heuristic and improving it at each iteration. First we denote $H = \{v_1,\ v_{i_2},\ \ldots,\ v_{i_r},\ v_{i_{r+1}},\ \ldots,\ v_{i_{k-1}},\ v_{i_k},\ v_{i_{k+1}},\ \ldots,\ v_n,\ v_1\}$ as a feasible solution for the original problem generated by a nearest neighbor heuristic. At each iteration we take a vertex v_{i_k} from its current position k and put it back in between vertices v_{i_r} and $v_{i_{r+1}}$ ($r < k$) getting a new tour H' with costs

$$z(H') = z(H) - s_r(v_{i_k})$$

where

$$s_r(v_{i_k}) = p(r)c_{i_r i_{r+1}} + L(v_{i_{r+1}}, v_{i_{k-1}}) + p(k-1)c_{i_{k-1} i_k} + p(k)c_{i_k i_{k+1}} - \left[p(r)c_{i_r i_k} + p(r+1)c_{i_k i_{r+1}} + p(k)c_{i_{k-1} i_{k+1}}\right]$$

and $L(v_i, v_j)$ is the length of the path that goes from v_i to v_j. So just the costs of the edges that are not used any more are replaced by the costs of the "new" edges.
Now we can choose a vertex $v_{i_{k^*}}$ and a position r^* at each iteration so that

$$s_{r^*}\left(v_{i_{k^*}}\right) = \max_{r,k} \left\{s_r\left(v_{i_k}\right)\right\}$$

Then we reposition $v_{i_{k^*}}$ between the vertices $v_{i_{r^*}}$ and $v_{i_{r^*+1}}$ if $s_{r^*}\left(v_{i_{k^*}}\right) > 0$. If this is not the case, the search for an upper bound is completed as there is no possibility any more to get a better solution.

3.3.3 Branch and Bound

Finally we can solve the problem using a classical branch and bound approach, with a simple depth-first tree search algorithm.

For each node m at level k there exists an f-path $F_m = \left\{v_{i_1},\ v_{i_2},\ \ldots,\ v_{i_{k-1}},\ v_{i_k}\right\}$ of cardinality $k-1$ with the starting point $v_{i_1} \equiv v_1$ ending at v_{i_k}. The cost of this path is given by

$$f\left(F_m\right) = p(1)c_{i_1 i_2} + p(2)c_{i_2 i_3} + \ldots + p(k-1)c_{i_{k-1} i_k}\ .$$

Now we denote $\overline{V} = V - F_m$ as the set of nodes, which have not yet been visited. If we want to branch forward from m at level k, we have to choose nodes out of $\left|\overline{V}\right|$, each generating an f-path of cardinality k

$$F_r = \left\{v_{i_1},\ v_{i_2},\ \ldots,\ v_{i_{k-1}},\ v_{i_k},\ v_{i_{k+1}}\right\} \qquad \forall\ v_{i_{k+1}} \in \overline{V}\ .$$

Then we can eliminate the node m from the tree search whenever

$$f\left(F_m\right) + h\left(n-k+1, v_{i_k}\right) - 2\sum_{i \in \overline{V}} l_i \geq z^*$$

with $z*$ as the cost of the best solution we were able to achieve up to this point.

3.3.4 Dominance Rules

To make the algorithm work better, we can discard more nodes from the tree search with the help of some dominance criterias.

3.3.4.1 Dominance Rule I

First we take two nodes m_1 and m_2 at level k of the tree corresponding to two f-paths F_{m_1} and F_{m_2} of cardinality $k-1$, which both end in the same vertex after visiting the same vertices. If

$$f(F_{m_1}) \leq f(F_{m_2})$$

this means that m_2 is dominated by m_1 and the node can be removed from the tree. So the cost of adding m_1 will be smaller than that of adding m_2 and we can therefore disregard m_2 from our decision.

3.3.4.2 Dominance Rule II

We denote an f-path $F = \{v_1, v_{i_2}, \ldots, v_{i_r}, v_{i_{r+1}}, \ldots, v_{j*}\}$ and insert the vertex $v_k \in \overline{V} = V - F$ between the vertices $v_{i_r}, v_{i_{r+1}} \in F$. If

$$p(r) c_{i_r} c_{i_{r+1}} > p(r) c_{i_r k} + p(r+1) c_{k i_{r+1}}$$

then F can not be part of the optimal solution as the costs of the tour with the integrated vertex v_k is smaller than that of the original one. Therefore the node on the tree corresponding to F can be eliminated.

3.3.4.3 Dominance Rule III

We denote an f-path $F = \{v_1,\ v_{i_2},\ \ldots,\ v_{i_r},\ v_{i_{r+1}},\ \ldots,\ v_{j^*}\}$ and a b-path $B = \{v_{j^*},\ v_{k_1},\ \ldots,\ v_{k_{s-1}},\ v_{k^*},\ v_{k_{s+1}},\ \ldots,\ v_1\}$ and try to insert the vertex $v_{k^*} \in \overline{V} = V - F$ between $v_{i_r}, v_{i_{r+1}} \in F$ getting $F^+ = \{v_1,\ v_{i_2},\ \ldots,\ v_{i_r},\ v_{k^*},\ v_{i_{r+1}},\ \ldots,\ v_{j^*}\}$. Naturally this will leave the corresponding b-path $B^- = \{v_{j^*},\ v_{k_1},\ \ldots,\ v_{k_{s-1}},\ v_{k_{s+1}},\ \ldots,\ v_1\}$ (see Figure 5). If

$$p(r)c_{i_r i_{r+1}} + L(v_{i_{r+1}}, v_{j^*}) + c_{j^* k^*} > p(r)c_{i_r k^*} + p(r+1)c_{k^* i_{r+1}}$$

then F can not lead to an optimal solution because its costs are higher than that of F^+ and we can eliminate the corresponding node on the tree.

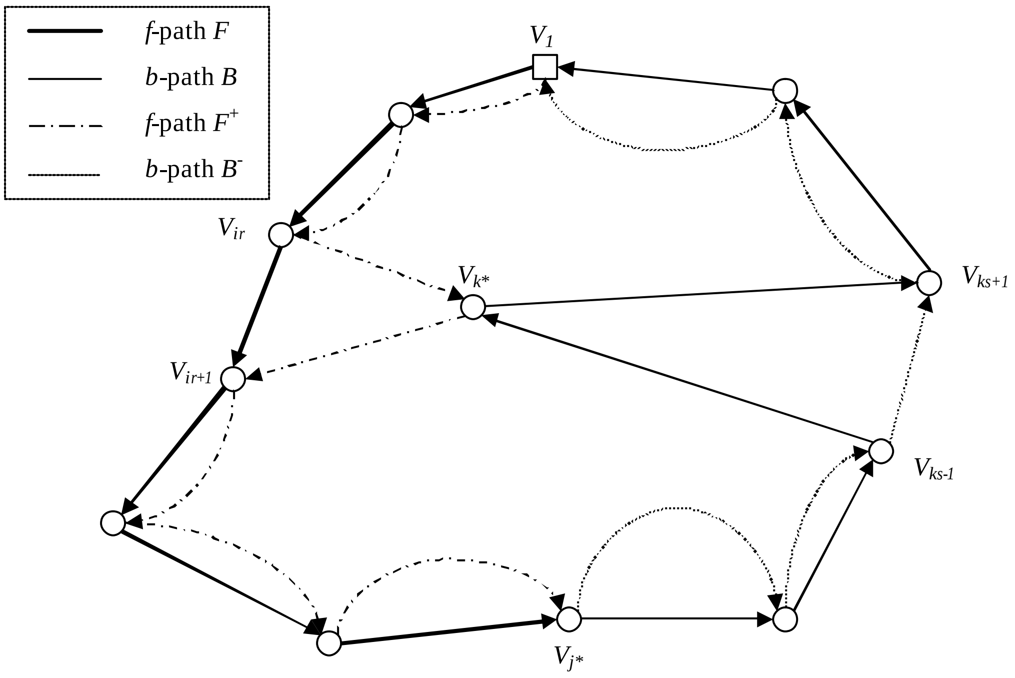

Figure 5: An example for a generated tour[9]

[9] Bianco et al., 1993, p. 85

3.4 Dynamic Programming Algorithm

The second algorithm Bianco et al. developed is a dynamic programming procedure which uses bounding functions to reduce the dimensionality of the state space graph.[10]

First we denote $C(S, v_i)$ as the minimum cost of an f-path with the usual starting point v_1 which visits every vertex of S and ends at a vertex $v_i \in S$. The dynamic programming recursion is

$$C(S, v_i) = \min_{v_j \in S - v_i} \left[C(S - v_i, v_j) + p(|S|) c_{ji} \right] \qquad \forall\ S \in V' = V - \{v_1\},\ \forall\ v_i \in S \tag{3.9}$$

The initial value is $C(\{v_i\}, v_i) = nc_{1i}$ for all $v_i \in V'$, which means that the cost of traveling from the starting point s to the first vertex has to be counted n times as it will influence the latencies of all the other customers.

At the end the repairman has to return to the origin so the optimal solution for this problem is given by

$$\min_{v_i \in V'} \left[C(V', v_i) + c_{i1} \right].$$

Additionally Bianco et al. introduced bounding criteria to increase the size of problems that can be solved by this method as the number of vertices of the state space group is too big for problems with $n > 15$.[11] All states satisfying the equation

$$C(S, v_i) + h(n - |S|, v_i) - 2 \sum_{i \in V - S} l_i \geq z_{UB}$$

can be eliminated.

[10] Bianco et al., 1993, pp. 86-87

[11] Bianco et al., 1993. p. 87

3.5 Dynamic Programming Heuristic

In the dynamic programming algorithm in Chapter 3.4 a rather high number of states are produced. To reduce them Bianco et al. introduced a heuristic which does not necessarily generate an optimal solution but is able to produce lower bounds that are often better than the lower bound LB created in Chapter 3.3.1.[12]

Equation (3.9) is able to generate the states (S, v_i) for increasing values of the cardinality of S. If states of cardinality $k+1$ are generated, that means to expand all states of cardinality k which quickly leads to an enormous number of states the algorithm can not handle in reasonable time. To avoid this we introduce a state space reduction.

First we denote $j_k = \{(S_1, v_{i_1}), (S_2, v_{i_2}), \ldots\}$ as the family of all states of cardinality k and $m_k = |j_k|$. For each state $(S_r, v_{i_r}) \in j_k$ we define a bound q_r^k for each solution in this state with $q_1^k \le q_2^k \le q_3^k \le \ldots \le q_{m_k}^k$.

$$q_r^k = C(S_r, v_{i_r}) + h(n - |S_r|, v_{i_r}) - 2 \sum_{v_i \in V - S_r} l_i$$

Now we introduce the constant $R_{\max}$, which is defined in advance, and eliminate from j_k all states (S_r, v_{i_r}) with $r = R_{\max}, \ldots, m_k$ and $q_r^k \ge q_{R_{\max}}^k$.

Then we define

$$q_{\min} = \min_{1<k\le n} \{ q_{R_{\max}}^k \mid m_k > R_{\max} \}$$

and let $\bar{z}$ be the value of the best solution the algorithm has found. Finally, we can state that if $\bar{z} > q_{\min}$, then $q_{\min}$ is a proper lower bound.

[12] Bianco et al., 1993, pp. 87-88

3.6 Computational Results

Bianco et al. developed a computer program to evaluate the algorithms described above.[13] They randomly generated coordinates for the vertices according to a uniform distribution in a 150 x 150 square and calculated the costs belonging to each arc with the Euclidean distance between the according vertices. They assumed a complete graph and introduced a time limit of 600 seconds to compute a solution.

Exact solutions for problems up to $n = 35$ vertices were produced with the help of the algorithm discussed in Chapter 3.3, while the one discussed in Chapter 3.4 failed once to find the optimum with $n = 35$ within the time limit. The second one is also a little bit faster when solving problems with more vertices.

For the dynamic programming heuristic (see Chapter 3.5) the algorithm was able to produce optimal solutions for problems up to $n = 25$ vertices and $R_{\max} = 200$, while for problems with a number of vertices between 30 and 35 and $R_{\max} = 400$ it was able to do that only six out of ten times. If there are more than 55 vertices to visit, the distance from optimality increases dramatically, but below this value it is at most 3%. It seems that with $n = 55$ vertices the limit of the algorithm has been reached.

[13] Biano et al., 1993, pp. 88-91

4 Variants of the TRP with One Repairman

This chapter gives a review of the variants of the TRP with exactly one repairman. Because of the characteristics of the TRP it can be used not only for the planning of a tour but also for a variety of other problems such as machine scheduling or in the area of computer networks. Therefore there exists a variety of articles which describe special forms of the TRP as the weighted TRP, the directed TRP or the on-line TRP. Some concentrate on different forms of the underlying graph, for example a path. And there are others which introduce additional constraints for the TRP like time windows or deadlines. Some of these variants will be descibed below.

4.1 The Line-TRP with and without Deadlines

The line-TRP is a one-dimensional version of the TRP, in which all vertices lie on a straight line (see Figure 6). This is a variant of the TRP that can be easily solved optimally because of the characteristics of the underlying graph.

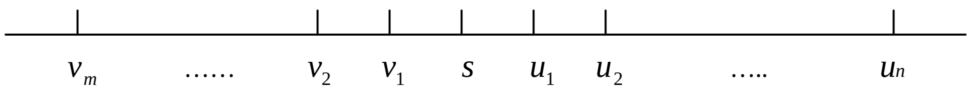

Figure 6: The line-TRP

Once more s represents the starting point and corresponds to v_0 and u_0 while $v_1, \dots, v_m$ are the customers on the left and $u_1, \dots, u_n$ on the right of the origin.

For the TSP the solution is rather simple. The salesman would just go straight ahead in one direction until he reaches the end and would then turn around to serve the customers on the other side of the origin. It is interesting to observe that the traveling repairman on the contrary may cross his own way several times, he will go back and forth to finish his tour in an optimal way (see Figure 7).

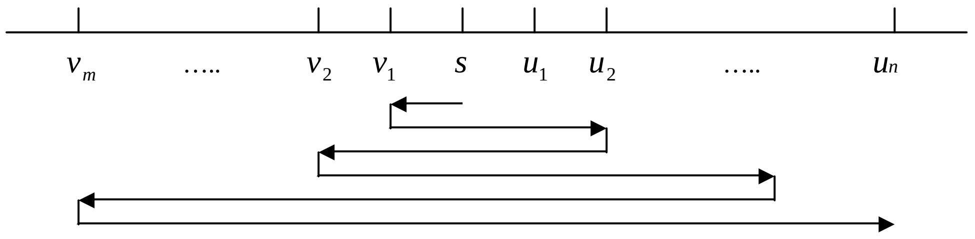

Figure 7: Possible tour for the line-TRP[14]

In 1986, Afrati et al. developed dynamic programming algorithms which are able to solve the line-TRP optimally with as well as without deadlines.

4.1.1 The Line-TRP without Deadlines

We determine that $[v_i, u_j]$ represents the state when the repairman is currently at point v_i and already visited all the vertices between v_i and u_j. We can assume this as it would not be optimal for the repairman to visit v_i without visiting v_{i-1} before. The repairman starts at $[v_0, u_0]$ and ends either at $[v_m, u_n]$ or $[u_n, v_m]$. It is also clear that in an optimal tour he will reach $[v_i, u_j]$ either from the state $[v_{i-1}, u_j]$ or $[u_j, v_{i-1}]$.

Now we let $c[v_i, u_j]$ denote the sum of the latencies of all customers already visited by the repairman. We could also say that it is the cost of reaching the state $[v_i, u_j]$. Following all these considerations we can introduce the following equations.

$$c[v_0, u_0] = c[u_0, v_0] = 0 \tag{4.1}$$

$$c[v_i, u_j] = \min \begin{Bmatrix} c[v_{i-1}, u_j] + (m+n+1-i-j)\ t[v_{i-1}, v_i], \\ c[u_j, v_{i-1}] + (m+n+1-i-j)\ t[u_j, v_i] \end{Bmatrix} \tag{4.2}$$

$$c[u_j, v_i] = \min \begin{Bmatrix} c[u_{j-1}, v_i] + (m+n+1-i-j)\ t[u_{j-1}, u_j], \\ c[v_i, u_{j-1}] + (m+n+1-i-j)\ t[v_i, u_j] \end{Bmatrix} \tag{4.3}$$

[14] Afrati et al., 1986, p. 81

with $t[v_i, u_j]$ representing the number of vertices the traveling repairman has not yet visited when being at state $[v_i, u_j]$. The explanation of equations (4.2) and (4.3) is the following: If we want to find out the minimum total delay of reaching a certain state we first have to compute which one of the two possible preceding states can be reached with a smaller total delay and allows us to complete the rest of the tour with a minimum of effort. Therefore we take the cost of reaching the preceding state and add the distance of reaching the last locations on the line multiplied by the number of unvisited locations at the preceding state.

Finally we can calculate the minimum cost of the optimal tour by

$$C = \min\{\, c[v_m, u_n], c[u_n, v_m]\}. \tag{4.4}$$

So we just have to record during each step of the algorithm which of the two possible preceding states provides the better value and reconstruct the optimal path. This dynamic programming algorithm is able to solve the line-TRP in $O(mn)$ time.[15]

4.1.2 The Line-TRP with Deadlines

Sometimes it is necessary for the traveling repairman to reach his customers at or before a given time. This could be the case because the customer is only available or the order must be completed until this point of time. Therefore we now introduce for each customer i a deadline d_i which indicates that the customer has to be served before this deadline expires. Unlike the version without deadlines the line-TRP with deadlines is NP-complete.[16]

Now that we do not only have to know where the traveling repairman is located but also at what time he is there we let $[v_i, u_j, t]$ denote the state that he is at customer v_i at time t and already served all the customers in the opposite

[15] Afrati et al., 1986, p. 81

[16] Afrati et al., 1986, p. 80

direction until u_j. This state has the cost $c[v_i, u_j, t]$ which we compute through the following equation.

$$c[v_i, u_j, t] = \begin{cases} \infty & if\ t > dv_i \\ \min \begin{Bmatrix} c[v_{i-1}, u_j, t - t[v_{i-1}, v_i]] + (m+n+1-i-j)\ t[v_{i-1}, v_i], \\ c[u_j, v_{i-1}, t - t[u_j, v_i]] + (m+n+1-i-j)\ t[u_j, v_i] \end{Bmatrix} & otherwise \end{cases} \quad (4.5)$$

In general this equation is the same as (4.2) but if the deadline of vertex v_i at which the repairman is located at time t has already expired, the cost will be set to infinite, so this subtour can not be feasible.

To calculate the optimum of the total cost the following equation needs to be solved.

$$C = \min \{ c[v_m, u_n, t], c[u_n, v_m, t] : t = 0, 1, \ldots, D \} \quad (4.6)$$

where D denotes the longest deadline of all customers.

This algorithm can solve the problem in $O(mnD)$ time.[17] Although Afrati et al. do not give a measurement for the accuracy of the algorithm they state that given a relative error at most e there exists a fully polynomial approximation scheme with the time bound $O(mn(m+n)/e)$.[18]

[17] Afrati et al., 1986, p. 87

[18] see footnote 17

4.2 The Weighted Line-TRP

The weighted line-TRP differs from the "ordinary" line-TRP only because of the weights $w_i \geq 0$, which are associated to each vertex v_i of the graph. This way the traveling repairman can make sure his tour takes into account that some customers are more important and do not have to wait too long for his visit. Another interpretation for the implementation of weights could be that the vertices are locations for machines that have to be repaired in which case the weights indicate the number of machines situated at each vertex.

In 2002, García et al. developed a linear algorithm that should solve this problem with the help of a Monge path-decomposable tridimensional array.[19]

The notations are

$s = v_0$	... the starting point
v_j	... the vertices on the left side of s with v_m as the vertex farthest away from s
u_k	... the vertices on the right side of s with u_n nearest s
$w(v_j) \geq 0$	... the weight of vertex v_j
T	... a feasible tour starting at s and visiting all points on the line
$c_T(s, v_j)$	... waiting time for vertex v_j in tour T
v_{j_1}	... the vertices v_j which are visited right after s before turning around where $0 \leq j_1 < j_2 < ... < j_h < j_{h+1} = m$
u_{k_1}	... the vertices u_k which are visited right after v_{j_1} before once more turning around where $n \geq k_1 > k_2 > ... > k_h = 1$

The objective is to minimize the latency of T according to the following equation.

$$L_T = \sum_{j=1}^{m} w(v_j)\ c_T(s, v_j) + \sum_{k=1}^{n} w(u_k)\ c_T(s, u_k) \rightarrow \min\ .$$

[19] García et al., 2002, pp. 27-29

Obviously the optimal tour could end either at v_m or at u_1 but it is reasonable to examine only the first possibility as the other one is calculated the same way and in the end one can choose the solution with the smaller costs.

We denote the tour $T = \{v_{j_1},\ u_{k_1},\ v_{j_2},\ u_{k_2},\ \ldots,\ v_{j_h},\ u_{k_h},\ v_{j_{h+1}} = v_m\}$ and start by solving the problem under the assumption that from all vertices u_i only u_1 has a positive weight $w(u_1) > 0$, then doing the same with $w(u_1) > 0$ and $w(u_2) > 0$ and so on until all vertices u_i have positive weights. $T(i)$ will represent the optimum path for $i = 0, 1, \ldots, n$, when all the points $u_{i+1}, u_{i+2}, \ldots, u_n$ have zero weight and the other keep their original weights, and E_i will be the latency of $T(i)$.

Now we define the sum of the weights of the vertices v_1 to v_j as $d(v_j) = \sum_{l=1}^{j} w(v_l)$ and the latency of u_k as $l(u_k) = \sum_{l=1}^{k} w(u_l)\ c(s, u_l)$ and assume that the vertices $u_n, \ldots, u_{k_1}$ have zero weight. Then the increment of latency of the vertices $v_{j_1+1}, \ldots, v_m$ is given by twice the distance from v_{j_1} to u_{k_1} (as we have to move first in the one, then in the other direction) times the sum of the weights of $v_{j_1}, \ldots, v_m$.

$$\Delta_1(j_1, k_1) = 2c(v_{j_1}, u_{k_1})\ [d(v_m) - d(v_{j_1})] \qquad \text{with } k_1 \le i$$

The increment of latency of the vertices $u_{k_i-1}, \ldots, u_1$ can be calculated by twice the distance from v_{j_1} to u_{k_1} times the weights of u_{k_i-1}, ..., u_1

$$\Delta_2(j_1, k_1) = 2c(v_{j_1}, u_{k_1})\ d(u_{k_1-1}).$$

Finally the latency of the vertices $u_{k_1},\ u_{k_1+1},\ \ldots,\ u_i$ is given by

$$\Delta_3(j_1,k_1,i)=\sum_{l=k_1}^{i}\left[2c\left(s,v_{j_1}\right)+c(s,u_l)\right]w_l\ =2c\left(s,v_{j_1}\right)\left[d(u_i)-d\left(u_{k_1-1}\right)\right]+l(u_i)-l\left(u_{k_1-1}\right).$$

It is also necessary to define that $l(u_0)=0$ and $d(u_0)=0$, as it is possible that $k_1=1$.

If we now substitute j_1 by $m-j$ and let be

$$a(i,j)=\Delta_3(m-j,k_1,i)+2c\left(s,v_{m-j}\right)d\left(u_{k_1-1}\right)+l\left(u_{k_i-1}\right)$$

and

$$b(j,k_1)=\Delta_1(m-j,k_1)+\Delta_2(m-j,k_1)-2c\left(s,v_{m-j}\right)d\left(u_{k_1-1}\right)-l\left(u_{k_i-1}\right)$$

then we finally have the following scheme of dynamic programming:[20]

$$E(i)=\min_{1\le j\le m}\ \min_{k_1\le i}\left\{E(k_1-1)+a(i,j)+b(j,k_1)\right\}\qquad i=1,\ 2,\ \ldots,\ n$$

Because the distances between the vertices and the weights are nonnegative numbers we can see that $A=\{a(i,j)\}$ and $B=\{b(j,k_1)\}$ are Monge matrices (see Chapter 7.5) and their entries can be calculated in constant time.

García et al. gave an algorithm which is able to solve this problem in $O(n+m)$ time in 1998.[21]

20 García et al., 2002, p. 29

21 García et al., 1998, pp. 3-9

4.3 The Directed TRP

In this chapter we examine the TRP whose underlying graph is directed, which means that $c(v,u)$, the cost of traveling from vertex v to vertex u, is not necessarily the same as the cost of crossing the edge in the opposite direction $c(u,v)$. In reality this could be necessary because there are one-way streets on the way and the repairman can not use the same route if he changes the direction.

In 2008, Nagarajan et al. developed an algorithm which starts by guessing break-points on the basis of the distances along the optimal path. Then the vertices will be split into sets situated between these break-points using a linear program and the algorithm calculates local tours for each set. Finally the sets will be connected again to a feasible tour and the minimum latency can be calculated.

We have a directed graph $G=(V,E)$ and the notations are

$s=v_0$	... the starting point
$u,\ v$	... vertices of the graph G
n	... number of vertices of G
$c^p(v,u)$	... the distance from v to u with $c^p(v,u)=\infty$, if u is not reachable from v
p	... the optimal tour starting at s and visiting all vertices v and u
$S\subseteq V$	... a subset of the vertex set V
$d^+(S)$	... the arcs leaving set S with $\{(v,u)\in E \mid u\in S,\ v\notin S\}$
$d^-(S)$	... the arcs entering set S with $\{(v,u)\in E \mid u\notin S,\ v\in S\}$
O^i	... the set of vertices which are visited between v_i and v_{i+1}
z^i	... the edge values corresponding to p restricted to O^i
$L=c(p)$	... the length of the optimal tour
y_u^i	... decision variable denoting if $u\in O_i$ ($y_u^i=1$) or not ($y_u^i=0$)

Additionally the number of vertices $l = \left\lceil \frac{1}{e} \right\rceil$, representing the break points as defined above, and the vertex v_i are introduced for which the following holds: for each $i = 1, 2, \ldots, l$, the vertex v_i is the last vertex with $c^p(s, v_i) \leq n^{ie} \frac{L}{n}$. Here e is a constant which can be chosen in advance and helps evaluate the performance of the algorithm (see below). If we let $F = \{v_0, v_1, \ldots, v_l\}$, then v_l will be the last vertex visited by p. The algorithm starts by guessing the length L and the number of vertices l.

Nagarajan et al. also present a theorem which states that in a Eulerian directed multi-graph $D = (U + s, A)$ there exists for each arc $f = (s, v) \in A$ an arc $e = (u, s) \in A$ so that it is possible to swap the arcs e and f with the arc (u, v) without destroying the directed connectivity between every pair of vertices in U.[22]

The mathematical formulation of the model is[23]:

$$\sum_{i=0}^{l-1} n^{(i+1)e} \frac{L}{n} \left(\sum_{u \notin F} y_u^i \right) \to \min \qquad (4.7)$$

subject to the following constraints:

$$z^i[d^+(u)] = z^i[d^-(u)] \qquad \forall\ u \in V \setminus \{v_i, v_{i+1}\}, \quad \forall\ i = 0, \ldots, l-1 \qquad (4.8)$$

$$z^i[d^+(v_i)] = z^i[d^-(v_{i+1})] = 1 \qquad \forall\ i = 0, \ldots, l-1 \qquad (4.9)$$

$$z^i[d^-(v_i)] = z^i[d^+(v_{i+1})] = 0 \qquad \forall\ i = 0, \ldots, l-1 \qquad (4.10)$$

$$z^i[d^-(S)] \geq y_u^i \qquad \forall\ \{u\} \subseteq S \subseteq V \setminus \{v_i\}, \quad \forall\ u \in V \setminus F, \quad \forall\ i = 0, \ldots, l-1 \qquad (4.11)$$

$$\sum_e c_e z^i(e) \leq n^{(i+1)e} \frac{L}{n} \qquad \forall\ i = 0, \ldots, l-1 \qquad (4.12)$$

[22] Nagarajan et al., 2008, p. 197

[23] Nagarajan et al., 2008, p.196

$$\sum_{i=0}^{l-1} y_u^i \geq 1 \qquad \forall\ u \in V \setminus F \qquad (4.13)$$

$$z^i(e) \geq 0 \qquad \forall\ arcs\ e, \quad \forall\ i = 0,\ \ldots,\ l-1 \qquad (4.14)$$

$$y_u^i \geq 0 \qquad \forall\ u \in V \setminus F, \quad \forall\ i = 0,\ \ldots,\ l-1 \qquad (4.15)$$

Constraint (4.8) makes sure that as many arcs enter the subset S as leave it while equations (4.9) and (4.10) guarantee that v_i is the first vertex that is visited in the subset S and v_{i+1} the first after leaving it. With constraint (4.11) we ensure that at least as many arcs are entering the subset S as there are vertices inside it, while inequality (4.13) states that at least one vertex has to be inside S. Constraint (4.12) ensures that the cost of using the arcs from vertices inside S to the starting and end point s should not be bigger than the costs of reaching v_{i+1}. Finally inequalities (4.14) and (4.15) represent the necessary non-negativity constraints for $z_i(e)$ and y_u^i. In the objective function (4.7) the optimal combination of vertices is determined.

For any $\frac{1}{\log n} < e < 1$ this algorithm has an approximation ratio of $O\left(r\frac{n^e}{e^3}\right)$ with r as the integrality gap for the asymmetric traveling salesman path problem for which Nagarajan et al. also developed an algorithm on the same basis. The time bound is given by $n^{O(1/e)}$.[24]

[24] Nagarajan et al., 2008, p. 196

4.4 The TRP on Weighted Trees

In this chapter we examine the TRP with a weighted tree T as the underlying graph, where the positive weights on its edges correspond to the distances between the vertices. In 1998, Goemans et al. developed a constant-factor approximation algorithm using solutions for the k-traveling salesman problem.[25] The idea behind this concept is that one first tries to find several shortest subtours which include at least k points of the vertex set $V = \{v_1, v_2, \dots, v_n\}$ for each value of k. These are finally concatenated to a tour which is rooted at the starting point v_1, visits all the vertices and therefore provides a feasible solution.

As stated above the algorithm will first solve the k-TSP on V for $k = 2, 3, \dots, n$ receiving tours $T_2, T_3, \dots, T_n$ with lengths $c_2 \leq c_3 \leq \dots \leq c_n$. We start with tour T_2 because T_1 simply consists of visiting the starting point v_1 at a cost of 0. Naturally we can not simply connect the so found subtours because each of them starts and ends at v_1 and some of the points would be visited more than once which can not lead to an optimal solution. Therefore we introduce an increasing set of indices $1 < j_1 < j_2 < \dots < j_m = n$ which will lead to the concatenated tour $T = T_{j_1}, T_{j_2}, \dots, T_{j_m}$. Starting from v_1 the repairman will cross T_{j_1}, afterwards T_{j_2} and so on until he traverses T_{j_m} and can finally return to the starting point.

For defining this tour the algorithm uses shortcuts to avoid visiting vertices which have already been visited. Additionally it determines the direction the subtour T_{j_i} has to be traversed to minimize the total latency of the vertices, which have not yet been visited in this subtour.

[25] Goemans et al., 1998, p. 114

Goemans et al. formulate this algorithm as follows:[26]

(i) For $k = 2, 3, \ldots, n$, compute T_k, the minimum-length k-TSP tour on V rooted at v_1. Let c_k denote the length of T_k.

(ii) Let G_n denote the complete graph on the vertex set $\{1, 2, \ldots, n\}$; turn G_n into a directed graph by orienting the edge (i, j) from $\min(i, j)$ to $\max(i, j)$.

(iii) Assign a length function to each directed arc of G_n; the length of arc (i, j) will be $\left(n - \frac{i+j}{2}\right)c_j$.

(iv) Compute the shortest $1-n$ path in G_n; suppose that it goes through vertices $1 = j_0 < j_1 < \ldots < j_m = n$.

(v) Output the concatenated tour $T = T_{j_1}, T_{j_2}, \ldots, T_{j_m}$.

The explanation for step (ii) and (iii) is as follows:
First all the edges of G_n are oriented in one direction and then costs are associated with them, so that the cost of the subtour T_{j_i} will be added to the latency of each of the following vertices, which have not yet been visited. Furthermore at most half of its cost contributes to the latency of the vertices which are first visited in this subtour. For this cost we can calculate as upper bound

$$\sum_i (n - j_i)c_{j_i} + \frac{1}{2}\sum_i (j_i - j_{i-1})c_{j_i} = \sum_i \left(n - \frac{j_{i-1} + j_i}{2}\right)c_{j_i} .$$

We can do this because we do not deduct the vertices appearing in a following subtour, which have already been visited before, so the optimal latency will be smaller than this term.

The approximation ratio of this model is $3{,}5912$[27] and it can be formulated as a linear program with $O(n)$ variables and $O(n^2)$ constraints.[28]

[26] Goemans et al., 1998, pp. 115-116

[27] Goemans et al., 1998, p. 113

[28] Goemans et al., 1998, p. 118

4.5 The Weighted TRP (WTRP)

As stated above in Chapter 4.2 the WTRP assigns weights to each vertex to indicate the importance of some customers or the urgency of visiting them as soon as possible. In this chapter we examine the WTRP when the underlying graph is a metric space.

In 2000, Wu developed an exact algorithm which solves the WTRP on a metric space through dynamic programming in polynomial time.[29] It consists of splitting off subtours and calculating the lengths of each of them. Then they are concatenated again in the order that guarantees an optimal solution.

We have a graph $G=(V,E)$ and the notations are[30]

s	... the starting point
u, v	... vertices of G
n	... number of vertices of G
$c_G(v,u)$	... the length of the shortest path from v to u on G
$w(v)$	... the weight of vertex v
$w(G)$	... the sum of the weights of all the vertices in G
P	... a subtour of G starting at s
r	... the vertex which is the connection between two subtours

If we have a subtour P of G we denote $L(P)$ as the weighted latency of P with

$$L(P)=\sum_{v\in V(P)} w(v)\ c_P(s,v)$$

and the weighted latency of P on G with

$$d(G,P)=L(P)+[w(G)-w(P)]\ c(P).$$

[29] Wu, 2000, pp. 225-228

[30] see footnote 18

As we can see $d(G,P)$ depends not only on the sum of the weighted latencies of the vertices which have already been visited but also on the weights of the unvisited ones.

First we define two subtours P_1 and P_2 with the same configuration, i.e. they both start and end at the same point and also visit the same vertices. Now we assume that $c(P_1) \leq c(P_2)$ and denote a third subtour P_0 which represents a complete tour if put together with either P_1 or P_2. We denote that by $Y_1 = P_1 // P_0$ we mean that P_0 will start at the vertex r, which is the vertex where P_1 ends, and that the connecting of the two subtours will generate a complete tour. If we let Y_1 be the tour obtained by concatenating P_0 and P_1, and Y_2 be the one obtained by concatenating P_0 and P_2, then $L(Y_1) \leq L(Y_2)$. Therefore we calculate the best subtour for every possible configuration starting with the one containing only the starting point and continuing with attaching one more vertex until we finally have a complete tour.

This algorithm is able to solve the problem in $O(n^2 2^n)$ time and is therefore very time consuming. In 2004, Wu et al. used it to develop an exact algorithm for the the classical TRP without weights.[31] For this purpose a pruning technique is presented that is very similar to a branch and bound algorithm as it uses upper and lower bounds to identify the subtours which can be eliminated.

The upper bound will be established through a simple greedy algorithm, which always searches the nearest vertex when deciding where to go next. The lower bound will be generated through a family of special functions. Both will be updated in every iteration.

Although the computing of the lower bounds is very time-consuming it is only done once in a preprocessing stage and will therefore not strain the time complexity of the entire algorithm too much. It will need $O(n^{k+1} + n^2 T)$ time to calculate the results where T is the number of generated subtours.

[31] Wu et al., 2004, pp. 303-309

4.6 The On-Line TRP (OL-TRP)

The OL-TRP is a special form of the TRP in which the number and locations of the requests which are released over time are not known in advance as this is the case in the offline version. This is a realistic assumption as in many applications the orders of the customers arrive over time.

An on-line algorithm is usually evaluated by comparing it with its off-line version. This is done by competitive analysis with the help of a competitive ratio. An algorithm is c-competitive if the cost of the on-line version is at most c times the cost of its optimal off-line counterpart.[32]

4.6.1 The Net Latency-OL-TRP (NL-OL-TRP)

In 2008, Allulli et al. analysed the NL-OL-TRP in a metric space which tries to minimize the net latency, the sum of the times the requests have to wait before being served.[33] If we introduce the release time t_i, which is the time at which the request s_i is announced, and the latency l_i of vertex v_i, then the net latency can be defined as

$$\sum_{i=1}^{n}(l_i - t_i) = \left(\sum_{i=1}^{n} l_i\right) - T$$

where T denotes the sum of the release times t_i. This makes sense as the waiting time of the customer actually begins when the request is made.

Throughout their work, Allulli et al. presented the proof that there exists no algorithm with a competitive ratio for the NL-OL-TRP in a metric space.[34]

[32] Allulli et al., 2008, p. 116

[33] Allulli et al., 2008, pp. 116-128

[34] Allulli et al., 2008, p. 118

4.6.2 The Weighted OL-TRP

In 2003, Krumke et al. developed two competitive algorithms for the weighted OL-TRP in a metric space.[35] In this place we will only discuss the one of them which delivers better results.

We assume that the traveling repairman is allowed to wait when there are no current requests that have to be served. He does not know how many requests there will come in or when the next one arrives. Obviously the requests can only be served after their release time.

As usual we have a graph $G=(V,E)$ and the notations are

s	... the origin of G
t_i	... the release time of request r_i
w_i	... the weight of request r_i
$c(v,u)$	... the distance from v to u
l_i	... the latency of v_i, i.e. the time v_i gets served

The objective is to minimize the weighted latency.

$$\sum_{i=1}^{n} w_i l_i \to \min$$

In the initialization phase the algorithm first searches for requests that have already been released at time 0. If there are none, the traveling repairman waits until t_1, when the first requests come in and we set $L := t_1$. Then the minimum time T at which the already released requests can be served is computed. When no further requests are released before T is reached, the repairman waits once more and we set $L := T$. If on the other hand some requests arrive at time t with $0 < t < T$, we set $L := t$. Additionally a random number $x \in]0,1]$ according to the uniform distribution gets chosen during the initialization phase.

[35] Krumke et al., 2003, pp. 279-294

Now the algorithm works in phases. First we set $B_0 := \frac{L}{2}$, $B_1 := 2^{-x}L$ and $B_i := 2^{i-1-x}L$ for every $i \geq 2$. For $i \geq 1$ the ith phase starts at time B_i, at which the algorithm searches for a way to satisfy the requests that already have come in but have not yet been scheduled with the following constraints:[36]

i. The schedule has to start and end in the starting point s.
ii. The length of the schedule should be at most $2B_i = B_{i+1}$.
iii. The weights of the requests which are served among all schedules should be maximized without violating (i) and (ii).

The calculated schedule has to be followed from time B_{i+1} until B_{i+2} according to the algorithm.

Because of constraint (ii) it is guaranteed that the tour computed by the algorithm can be finished before the next phase starts while constraint (iii) ensures that requests with large weights are favored.

Krumke et al. do not give a competitive ratio for the OL-TRP, but for the on-line dial-a-ride problem, of which the OL-TRP is a special case. The on-line dial-a-ride problem also minimizes the weighted latency but has a few more restrictions. First the traveling repairman has a certain capacity C because he needs to deliver items. Therefore each request not only has one position at which it has to be served, but a source and a destination between which the objects have to be transported. For this problem the above described algorithm is c-competitive with $c = \frac{4}{\ln 2} \approx 5.7708$.

The competitive analysis is often criticized because it concentrates on the on-line versions of the problems. Therefore the input instance is usually generated in a way that can discriminate the off-line version, while the on-line adversary can serve it rather effectively.[37]

[36] Krumke et al. 2003, p. 287

[37] Allulli et al., 2008, p. 118

5 The TRP with Multiple Repairmen (The k-TRP)

The k-TRP tries to solve the problem when there is a number of k repairmen available to fulfill the requests. Each one starts at the depot s and makes his tour, and all together they have to serve each of the n customer requests. These tours have to be disjoint except for s because obviously it would not be optimal if one customer would be served by more than one repairman. As before the goal is to minimize the average time the customers have to wait until their requests have been answered.

This problem has first been examined by Fakcharoenphol et al. in 2007 when they developed a polynomial-time approximation algorithm to solve it.[38] Below we will examine it in more detail.

As the k-TRP is a variant of the TRP there also exist a lot of special forms with differing underlying graphs or additional constraints. In the next chapters we will look more closely at the k-TRP with repairtimes and the on-line k-TRP.

[38] Fakcharoenphol et al., 2007, pp. 40:1-40:16

5.1 The k-TRP

In 2007, Fakcharoenphol et al. introduced an algorithm for the k-TRP in a metric space using the concept of the i-Minimum Spanning Tree problem (i-MST problem) and improved it with the help of the i-stroll problem.[39] The i-MST problem consists of finding the least expensive tree starting at s and visiting exactly i vertices (see Chapter 7.3), while the i-stroll problem finds the least expensive path with the same requirements. Both problems are NP-hard, so we can only use approximation algorithms for this technique.

First we give a short problem description. If we have a tour S visiting the vertices $s = v_0,\ v_1,\ \ldots,\ v_m$ and distances $c(v_j, v_{j+1})$ from v_j to v_{j+1}, then the cost of S is given by

$$\sum_{i=0}^{m} l_i \text{, where } l_i = \sum_{j=0}^{i-1} c(v_j, v_{j+1}).$$

Because there are k tours we have to summarize the costs of all of them to calculate the final result.

Now we describe a subroutine which gives back an i-MST, a tree of cost at most aB spanning at least i vertices.

Subroutine $BudgetTree\ (B)$:[40]

```
$T = [empty\ tree]$
for $i = 1$ to $n$ do:
        if $M(i)$ has cost at most $aB$ then
                $T = M(i)$
return $T$
```

[39] see footnote 27

[40] Fakcharoenphol et al., 2007, p. 40:6

where $M(i)$ is an a -approximation algorithm covering i vertices for the i-MST and B is a constant previously fixed.

The result of this subroutine depends heavily on the ability to produce some good i-MSTs. If the approximation ratio a is too high, then the computed $BudgetTree$ will influence the result of the whole algorithm negatively.

Now we introduce the constant b, for which Fakcharoenphol et al. give the optimal value of $b = 1{,}616$[41], and the random variable U, which is selected by the uniform distribution on $[0,\ 1]$ and can describe the following algorithm for solving the problem.

Algorithm:[42]

```
Choose $c = b^U$ according to the random variable $U$, which has a uniform
distribution on $[0,\ 1]$. For each $j \geq 0$:
    For $l = 1,\ \ldots,\ k$
        Let $T_l^j = BudgetTree\ (cb^j)$. Remove the vertices of $T_l^j$ other than $s$
        from the graph.
    Arbitrarily give each repairman one of the $k$ trees from this stage.
Tell each repairman to traverse his trees in increasing order of $j$, and to
traverse each tree either in the forward Euler tour direction or the backward
Euler tour direction according to the flip of an unbiased coin.
```

This algorithm computes trees of cost at most $a\ c$, $a\ cb$, $a\ cb^2$, ... covering as many vertices as possible. Every $BudgetTree$ gives back a set of trees for each of the k repairmen with nearly exponentially increasing length (as $c = b^U$), which they have to traverse without visiting a vertex more than once.

[41] Fakcharoenphol et al., 2007, p. 40 :

[42] Fakcharoenphol et al., 2007, p. 40:12

This algorithm has an approximation ratio of $8.497a$ [43], but can be replaced by using the i-stroll instead of the i-MST with an approximation ratio of 8.497 as Chaudhuri et al. showed in 2003.[44]

[43] Fakcharoenphol et al., 2007, p. 40:15

[44] Chaudhuri et al., 2003, pp. 36-45

5.2 The k-TRP with Repairtimes (GKTRP)

In all the models already discussed we assumed that there are no repairtimes for the customers. Now we would like to examine the k-TRP with variable repairtimes associated to each vertex (GKTRP). This is a realistic assumption because the repairman may have to spend some time at the customer's house in order to satisfy his request.

In 2006, Jothi et al. presented approximation algorithms for the GKTRP with variable and uniform repairtimes respectively.[45]

5.2.1 The GKTRP with Non-Uniform Repairtimes

First we would like to examine the GKTRP with different repairtimes associated to each customer.

We have an undirected graph $G = (V, E)$ and the notations are

s	... starting point
$u,\ v$	... vertices of graph G
$c(u,v)$	... distance between u and v
r_i	... repairtime of vertex v_i
$l(v_i)$	... latency of vertex v_i
t_k	... one of the k tours which cover together all the vertices of G

First we denote $M = \{v_1,\ v_2,\ \ldots,\ v_k\}$ as the set of vertices which have the k largest repairtimes and as G' the graph G without these vertices. Then we change the graph G' such that we add half of the repairtimes of v_i and v_j to $c(v_i, v_j)$, set all the repairtimes r_i to 0 and introduce the new graph as $G*$ (see Figure 8).

[45] Jothi et al., 2006, pp. 294-303

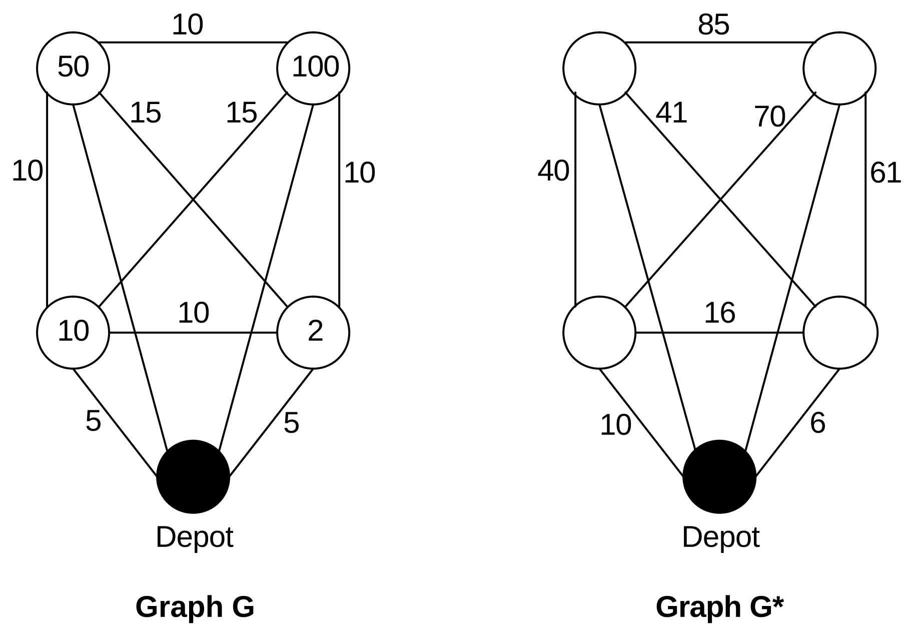

Figure 8: Original graph G and transformed graph G*[46]

Now the procedure will be to seek a b-approximate solution for $G*$ using the best algorithm currently known to solve the k-TRP without repairtimes and get as result the set of tours $t_1,\ t_2,\ \ldots,\ t_k$. We transfer the so found tours into G', so that they visit the same vertices in the same order and can calculate the sum of the latencies of the customers apx' by

$$apx' = b\ \ opt* - \sum_{i \in V \setminus \mathrm{M}} \frac{\mathrm{r_i}}{2}$$

where $opt*$ is the optimal solution for the k-TRP in $G*$.

This can be easily explained as in $G*$ half of the repairtimes of v_i have been added to the distance between its predecessor and itself and is therefore part of the latency of v_i. If we subtract this amount once more we get the latency of this vertex in G'.

[46] Jothi et al., 2006, p. 294

Finally we add to each tour t_i a vertex v_i out of M for all i and get a set of k feasible tours in the original graph G. The sum of the latencies of the customers in G is then given by

$$apx = apx' + \sum_{i=1}^{k} l(v_i)$$

which is the optimal solution of G' plus the latencies of the customers which have previously been subtracted from the original graph.

The approximation ratio of this algorithm is $\left(\frac{3}{2}b + \frac{1}{2}\right)$, which depends heavily on the algorithm used to create the optimal tour for $G*$.[47] The currently best achievable approximation ratio for the k-TRP without repairtimes has been reached by Chaudhuri et al. and is given by $b = 8{,}49$[48] (see Chapter 5.1) leading to an approximation ratio of $13{,}235$ for this algorithm.

5.2.2 The GKTRP with Uniform Repairtimes

In this chapter we examine the special case when the repairtimes are all the same for each vertex v_i. Jothi et al. gave two approximation algorithms to solve this problem, one with an approximation ratio which decreases with increasing $\frac{k}{n}$ (algorithm 1), while the approximation ratio of the other increases with $\frac{k}{n}$ (algorithm 2).[49] Depending on this factor one can choose which algorithm to use to get an approximation ratio as small as possible.

5.2.2.1 Algorithm 1

The first algorithm works on a case-by-case basis meaning that there are different procedures depending on the value of k. The notations are the same as in the model with non-uniform repairtimes, but we build the new graph $G*$

[47] Jothi et al., 2007, p. 298

[48] Chaudhuri et al., 2003, p. 38

[49] Jothi et al., 2007, pp. 299-301

directly out of G, once more adding half of the repairtimes of v_i and v_j to $c(v_i, v_j)$ and setting all repairtimes r_i to zero. Then we arrange the vertices v_i according to their distances to the starting point, so that $c(s, v_1) \leq c(s, v_2) \leq \ldots \leq c(s, v_n)$.

Case 1: $k \geq \frac{n}{2}$

In this case we assign to each repairman i the vertex v_i for all $i \leq k$ and let the repairmen 1 to $n-k$ visit a second one of the remaining vertices. This way it is ensured that customers with smaller distances to the origin are visited earlier and the latencies of the remaining customers do not get too high.

The approximation ratio of the algorithm in this case is at most 2.

Case l: $\frac{n}{l+1} \leq k < \frac{n}{l} \qquad \forall\ l > 1$

In this case the assignment of vertices to the k repairmen works the same way as above. First each repairman i visits the vertex v_i, then one of the vertices $\{v_{k+1},\ v_{k+2},\ \ldots,\ v_{2k}\}$ and so on until all the vertices have been visited.

Here the approximation ratio depends on the value of l, so for values of $l = 3,\ 4,\ 5$ the ratios are $4.83,\ 6.43,\ 8.1$.[50]

5.2.2.2 Algorithm 2

This algorithm works in the same way as the one examined in Chapter 5.2.1 but without eliminating the set of vertices M to get the graph G'. We create $G*$ directly out of G by once again adding half of the repairtimes of v_i and v_j to $c(v_i, v_j)$ and setting r_i to 0. As before we then search for a b-approximate solution to $G*$ using the best known procedure for finding a solution for the k-TRP.

[50] Jothi et al., 2007, p. 301

For this algorithm the b -approximation ratio is given by $b + \left(\frac{b - 1}{\frac{n}{k} - 1} \right)$. Once more assuming $b = 8{,}49$, this leads to an approximation ratio of at most $10{,}2283$ for $k \geq 0.188364n$ which decreases with an increasing value of $\frac{n}{k}$. If k is smaller than $0.188364n$, then it will be better to use algorithm 1 to solve the problem. [51]

[51] Jothi et al., 2007, p. 301

5.3 The On-Line k-TRP (OL-KTRP)

The OL-KTRP is a special form of the k-TRP where the repairmen do not know in advance when there will be a request of a customer. As in Chapter 4.6 the requests will be released over time and have to be served as quickly as possible to minimize the average latencies experienced by the customers. But this time we have k repairmen to satisfy them.

In 2006, Bonifaci et al. presented a c-competitive algorithm for the OL-KTRP which first divides the k repairmen into groups of $k*$ repairmen, where the value of $k*$ is given in advance.[52] At a previously defined point of time B_i one group starts to serve the requests already released while the others wait at the origin. Some time later the next group starts their tours and so on. Bonifaci et al. call their algorithm Group Interval.

Algorithm Group Interval[53]

Divide the servers into $g = \left\lfloor \frac{k}{k*} \right\rfloor$ disjoint sets (groups) of $k*$ servers each. Any remaining server is not used by the algorithm.

Let L be the earliest time that any request can be completed (wlog $L > 0$). For $i = 0,\ 1,\ \ldots$, define $B_i = a^i L$ where $a = 3^{\frac{1}{g}}$.

At time B_i, compute a set of paths $S_i = \{P_1^i,\ \ldots,\ P_{k*}^i\}$ for the set of yet unserved requests released up to time B_i with the following properties:

(i) every P_j^i starts at the origin s;

(ii) $\max_j \left| P_j^i \right| \le B_i$;

(iii) S_i maximizes the number of requests served among all schedules satisfying the first two conditions.

Starting at time B_i, the j-th server in the (i mod g)-th group follows path P_j^i, then returns to s at full speed.

---

[52] Bonifaci et al., 2006, pp. 87-89

[53] Bonifaci et al., 2006, p. 88

As stated above the k repairmen get divided into g groups. Then the constant a gets introduced which decreases with an increasing number of groups. As the algorithm computes schedules at time B_i (depending on a), then at B_{i+1} and so on, the repairmen have the period of $B_{i+1} - B_1$ to serve all the already released but not yet visited customers. This period gets smaller with an increasing number of groups as the lengths of the tours get shorter. After all the requests have been satisfied, the repairmen return to the starting point and wait for the release of new ones.

The competitive ratio c for this algorithm is given by $2 \cdot 3^{\frac{1}{g}}$.[54]

[54] Bonifaci et al., 2006, p. 88

6 Applications of the TRP

Because of the structure of the TRP there exist a lot of applications for the TRP. Many different problems can be described and calculated by using the model or at least part of it. Naturally the most obvious group of problems is that of the delivery problems, for example the pizza delivery, where one puts together several orders and has to find a tour which minimizes the average arrival time at the customers so that the pizzas will not get cold in the meantime. Another application would be that of finding a route for automated guided vehicles through the cells of a flexible manufacturing system. One can also interpret certain scheduling problems as TRPs (see Chapter 6.2).

In the field of computer networks one can also find applications for the TRP, for example in the area of diskhead scheduling, where the objective is to minimize the wasted time by hard disk seeks. The model of the TRP can also be used when one tries to find a certain information which is located somewhere in the network.

Below we will examine two possible applications at more detail, first an emergency vehicle dispatching system for an electric utility company in Chile, then a technician and task scheduling problem.

6.1 An Emergency Vehicle Dispatching System

In this chapter an example of an application of the TRP will be described at more detail. We will especially examine the factors which have to be taken into account when solving problems in reality.

Chilectra is the electric utility responsible for Santiago, the capital of Chile, and incorporates a special emergency services division which takes care of electrical breakdowns in the metropolitan area. In 1999, Weintraub et al. developed a computerised system helping to organize the dispatching of vehicles to the emergency locations served by the emergency unit responsible for three out of the sixteen municipalities in Santiago.[55] Before an operator decided which vehicle served which emergency in which order only with the help of his experience and intuition.

The geography of the affected part of the city determines the underlying graph needed to describe the problem. The nodes of it represent areas of two to five blocks and the distances between them are calculated with the help of the actual travel times under traffic scenarios depending on the time of day. In order to predict future emergencies a model was created using an exponential smoothing approach based on experience with earlier breakdowns. Additionally a priority factor is assigned to each emergency which can be interpreted as a weight.

The notations of the model developed by Weintraub et al. are

I	... the set of the already known breakdowns
J	... the set of zones
P_i	... the weight of the priority factor for breakdown i
T_i	... the service time including the time needed to reach breakdown i
F_j	... the expected number of breakdowns for zone j
K_{jt}	... the penalty factor depending on the distance to zone j in time t

[55] Weintraub et al., 1999, pp. 690-696

The objective function developed by Weintraub et al. is[56]

$$a \sum_{i \in I} P_i T_i + b \sum_{t} \sum_{j \in J} F_j K_{jt} \rightarrow \min$$

The first term of the equation can be interpreted as the already familiar sum of waiting times of the customers times the correspondent prioritiy levels, i.e. the weights. The second term takes the future demands into account. The parameters a and b determine the weight of each of the two terms.

Now we will take a closer look at some parameters which are part of the model and some of the factors which influence the decision in particular.

6.1.1 The Weights a and b

The weights a and b can be interpreted as the service quality for the breakdowns that are already known and the penalty that is imposed on the vehicle if it is located far away of future breakdowns. Weintraub et al. chose to define the values of the two weights through a process of simulation to get adequate solutions in all likely events and then validated them to maximize the level of global service quality.

6.1.2 The Weight of Priorities P_i

The priorities P_i of the breakdowns are defined by Chilectra. There are five priority levels with weights which do not grow linear (see Table 2). A critical breakdown that could also endanger humans would be classified as priority 1, while a domestic loss of power would be of priority 5. Additionally a breakdown will move up a level every 30 minutes if it has not been served until this time, so that it is ensured that no emergency will be neglected for a long time.

[56] Weintraub et al., 1999, p. 691

Priority	**Weight** P_i
1	10
2	5
3	3
4	3
5	1

Table 2: Weighted factors P_i according to breakdown priorities

6.1.3 The Forecast for Daily Demands

To predict future breakdowns effectively the development in previous years must be examined. First one can see that there exists a rather strong seasonality effect, so that in winter an increase in the number of breakdowns due to more wind and rain can be observed while in the summer months they become less because people tend to leave the city for vacation. At the daily level the emergency calls decrease during the night hours while they increase during the evening. Additionally there is a rise of breakdowns noticeable throughout the years because of the growth of the population and the construction of new houses. All these factors have to be taken into account when defining the future demand.

6.1.4 The Implementation of the Algorithm

Weintraub et al. used an approach to implement the algorithm where they first grouped the requests together according to the geographical sector they were situated in and then assigned vehicles to these groups. Now they defined the tours for each of them by adding only one node at a time which they took out of a predefined group of neighbors. Before inserting a new node, all possible combinations of sequences including this node are tested to find the one that leads to minimal cost. Additionally Weintraub et al. had to make sure that requests with high priority have to be visited early. Finally they improved the solution by using a load balancing approach where they compared the workload of each vehicle and tried to balance it.

The model of Weintraub et al. proved to be rather efficient in the testing phase. It resulted in an improvement of 16% in response time and even 53% during rainy days when the number of breakdown increased significantly.[57]

[57] Weintraub et al., 1999, p. 690

6.2 A Technician and Task Scheduling Problem (TTSP)

In the TTSP we have a set of tasks which have to be fulfilled by technicians with specified skills. The level of a technician in a skill is given by an integer from 0 to p, with 0 meaning that he has no knowledge at all in this area. The tasks to be completed require different skill levels and some additionally have to be fulfilled by more than one technician. The problem consists of assigning to each task the necessary technicians with convenient skill levels in a most effective way. For this purpose technicians are grouped together according to their skill levels and have to stay together for all day. It is also possible that some technicians are not available on some days.

An additional constraint is given by the fact that some tasks cannot be completed before others are performed, so that each task has a set of predecessors and a set of successors. All tasks have given values for the duration it takes to fulfil them, the outsourcing cost, for which a certain budget is available, and a priority level, where a weight is assigned to each priority level.

In 2008, Cordeau et al. developed a heuristic to solve the above described problem.[58]

The notations are

N	... the set of tasks with N_p as the set of tasks with priority p and N^s as ... the set of tasks with successors
T	... the set of technicians with T_k as the set of available technicians at day k
s_i	... the set of successors of task i
d_i	... the time required to fulfill task i
c_i	... the cost of outsourcing task i
C	... the budget for outsourcing costs
p_i	... the priority level of task i with $p_i \in \{1, 2, 3, 4\}$

[58] Cordeau et al., 2008, pp. 1-25

w_i	... the weight according to the priority level of task i with $w_i \in \{28,\ 14,\ 4,\ 1\}$
x_{jkr}	... the binary variable indicating whether technician j is part of the team r on day k or not
y_{ikr}	... the binary variable indicating whether task i is assigned to team r on day k or not
z_i	... the binary variable indicating whether task i is outsourced or not
$u_{ii'}$	... the binary variable indicating whether task i has been fulfilled before task i' starts
e_p	... the time when the last task of priority p has been fulfilled
b_i	... the starting time of task i
s^i_{ab}	... element of the skill requirement matrix indicating the number of technicians with a skill level of at least a in the domain b required to fulfill the task i
v^j_{ab}	... element of the skill matrix indicating the skill level a in the domain b of technician j
M	... a large number

The objective is to minimize the weighted makespan of each priority level, so it is given by[59]

$$\sum_{p=1}^{4} w_p e_p \to \min \ . \tag{6.1}$$

This is the same objective function as that of the weighted TRP.

[59] Cordeau et al., 2008, p. 5

The constraints are the following[60]

$$e_p \ge b_i + d_i \qquad \forall\ p \in \{1,\ 2,\ 3\},\ \forall\ i \in N_p \tag{6.2}$$

$$e_4 \ge b_i + d_i \qquad \forall\ i \in N \tag{6.3}$$

$$\sum c_i z_i \le C \tag{6.4}$$

$$|s_i| z_i \le \sum_{i' \in s_i} z_{i'} \qquad \forall\ i \in N^s \tag{6.5}$$

$$\sum_{r=1}^{m} x_{jkr} \le 1 \qquad \forall\ k \in K,\ \forall\ j \in T_k \tag{6.6}$$

$$\sum_{r=1}^{m} x_{jkr} = 0 \qquad \forall\ k \in K,\ \forall\ j \in T \setminus T_k \tag{6.7}$$

$$z_i + \sum_{k \in K} \sum_{r=1}^{m} y_{ikr} = 1 \qquad \forall\ i \in N \tag{6.8}$$

$$y_{ikr} s_{ab}^{i} \le \sum_{j \in T_k} v_{ab}^{i} x_{jkr} \qquad \forall\ i \in N,\ \forall\ k \in K,\ \forall\ r \in \{1,\ \dots,\ m\},\ \forall\ a \in \{1,\ \dots,\ p\},\ \forall\ b \in \{1,\ \dots,\ q\} \tag{6.9}$$

$$b_i + d_i \le b_{i'} + M z_i \qquad \forall\ i \in N^s,\ \forall\ i' \in s_i \tag{6.10}$$

$$120\ (k-1) \sum_{r=1}^{m} y_{ikr} \le b_i \qquad \forall\ i \in N,\ \forall\ k \in K \tag{6.11}$$

$$120\ k \sum_{r=1}^{m} y_{ikr} \ge b_i + d_i \qquad \forall\ i \in N,\ \forall\ k \in K \tag{6.12}$$

$$b_i + d_i - (1 - u_{ii'}) M \le b_{i'} \qquad \forall\ i,\ i' \in N,\ i \ne i' \tag{6.13}$$

$$y_{ikr} + y_{i'kr} - u_{ii'} - u_{i'i} \le 1 \qquad \forall\ i,\ i' \in N,\ i \ne i',\ \forall\ k \in K,\ \forall\ r \in \{1,\ \dots,\ m\} \tag{6.14}$$

$$x_{jkr} \in \{0,\ 1\} \qquad \forall\ j \in T,\ \forall\ k \in K,\ \forall\ r \in \{1,\ \dots,\ m\} \tag{6.15}$$

$$y_{ikr} \in \{0,\ 1\} \qquad \forall\ i \in N,\ \forall\ k \in K,\ \forall\ r \in \{1,\ \dots,\ m\} \tag{6.16}$$

$$z_i \in \{0,\ 1\} \qquad \forall\ i \in N \tag{6.17}$$

$$u_{ii'} \in \{0,\ 1\} \qquad \forall\ i,\ i' \in N,\ i \ne i' \tag{6.18}$$

$$e_p \ge 0 \qquad \forall\ p \in \{1,\ 2,\ 3,\ 4\} \tag{6.19}$$

$$b_i \ge 0 \qquad \forall\ i \in N \tag{6.20}$$

[60] see footnote 59

The constraints (6.2) and (6.3) define the ending time of all tasks of priority p. Inequalities (6.4) and (6.5) ensure that no more than the available budget is used for outsourced tasks and all successors of an outsourced task are outsourced, too. Restrictions (6.6) and (6.7) state that every technician is only part of one team per day and that no unavailable technician is used. Equality (6.8) ensures that every task is either fulfilled by a technician or outsourced while (6.9) assigns to each task that is not outsourced a team with necessary skills. Constraint (6.10) makes sure that a task is fulfilled before its successors start. Restrictions (6.11) and (6.12) introduce a lower and an upper bound for the starting time of the tasks. Inequality (6.13) ensures that if task i' should be fulfilled after task i then the starting time of i' has to be after the starting time of i, while (6.14) states that if both tasks are performed by the same team on the same day, they have to be fulfilled one after the other. Finally restrictions (6.15) to (6.18) introduce the binary variables and (6.19) and (6.20) define the nonnegativity constraints.

Cordeau et al. developed a construction heuristic to build teams and assign tasks to them and then an adaptive large neighborhood search heuristic (ALNS) which is able to further improve the initial solution. The construction heuristic first chooses seed tasks according to predefined criteria and builds teams to fulfill them. Then further tasks are assigned to the selected teams. The ALNS heuristic finally tries to improve the so found solution by destroying and afterwards repairing it. This means that first a subset of tasks is removed from the solution and then it will be reinserted again. For this purpose five different destroy and two repair heuristics are implemented into the algorithm.

As the above described heuristics have been developed for a challenge the test instances were predefined. They varied from 5 to 800 tasks with 5 to 150 technicians available. The ALNS heuristic achieved the second place by differing on average only 5,9% from the best known solutions for the test instances.

7 Basis and Solving Methods for the Traveling Repairman Problem

This chapter gives a description of the basis and the most common solving methods used for examining the TRP.

7.1 Branch and Bound[61]

The branch and bound technique is used for generating optimal solutions for optimization problems. It was first introduced by A. H. Land and A. G. Doig in 1960, who used it in the field of linear programming.

First the set of candidates S is split into several smaller sets $S_1, S_2, \dots$ which will lead to a tree structure (the search tree) with nodes as the corresponding subsets of S (branching). Then upper and lower bounds for each subset are generated (bounding). Now the bounds for the tree nodes are compared to each other and the "worse" ones get eliminated from the tree search (pruning). If there is only one candidate left or an upper bound for the set of candidates matches the lower bound, the recursion finally stops.

7.2 Dynamic Programming[62]

The concept of dynamic programming in its actual form was developed by Richard Bellman in 1953. It consists of breaking down a complex problem into simpler subproblems, which means splitting it up into several decision steps that can be solved more easily. Dynamic programming can only be used as a solving method when the problem has two specific properties: the overlapping subproblems and the optimal substructure.

A problem has an optimal substructure if it can be broken down recursively, which means that by solving its subproblems over time the original problem can

[61] http://en.wikipedia.org/wiki/Branch_and_bound; 05.06.2009

[62] http://en.wikipedia.org/wiki/Dynamic_programming; 18.08.2009

be solved. If these subproblems can be reused multiple times then it also has the property of overlapping subproblems.

7.3 Graph Theory[63]

As the TRP is usually described on a graph a short overview over graph theory is given in this chapter.

A graph $G=(V,E)$ has a set of n vertices $V=\{1,\ 2,\ \ldots,\ n\}$ and an edge set $E \subseteq V \times V$. Every vertex represents a customer the traveling repairman has to visit. For each edge (i,j) there are known costs c_{ij} which can be interpreted as the distance between the vertices i and j.

The graph can be directed or undirected. A graph is directed if the TRP is asymmetric and all edges are pointing in one direction, so they can be represented by an arrow (see Figure 9). The traveling repairman is only allowed to travel from i to j but not from j to i.

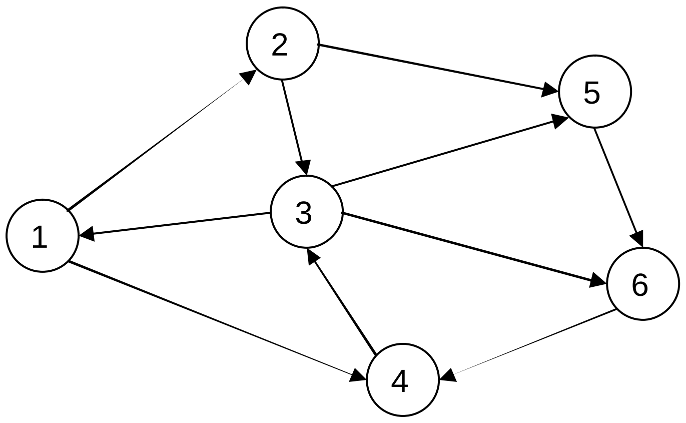

Figure 9: A directed graph

A graph is undirected if the TRP is symmetric and all edges are undirected (see Figure 10). So the traveling repairman is allowed to travel from i to j and also the other way round.

[63] http://en.wikipedia.org/wiki/Graph_(mathematics)#Undirected_graph; 05.06.2009

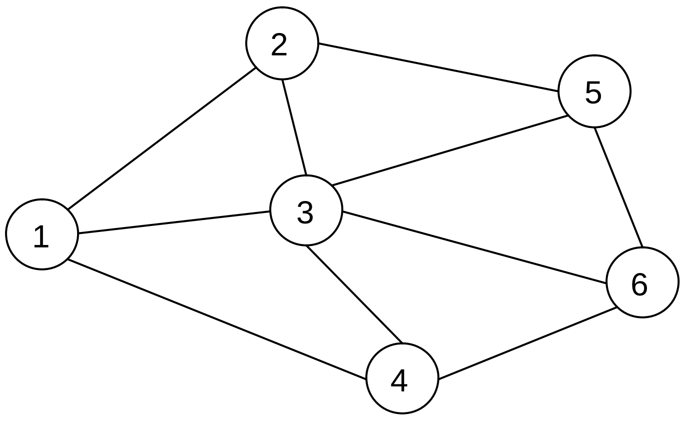

Figure 10: An undirected graph

A Hamiltonian circle is a path which visits each vertex exactly once and then returns to the starting point (see Figure 11). It is the graphic equivalent to the tour the traveling repairman has to make.

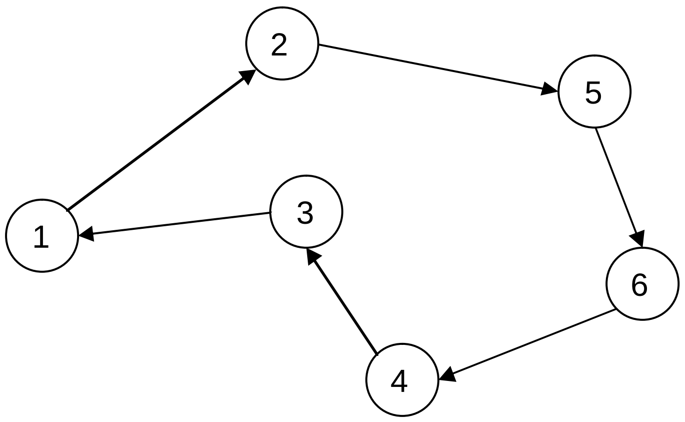

Figure 11: A Hamiltonian circle

A spanning tree of an undirected graph connects all the vertices without forming a cycle (see Figure 12). The spanning tree with the smallest costs is called the minimum spanning tree (MST).

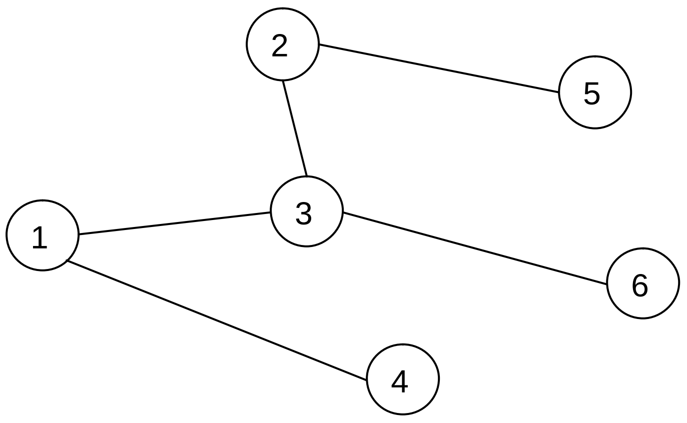

Figure 12: Spanning tree

A Eulerian graph is a graph which allows to construct a Eulerian circuit, i.e. it is possible to use each edge exactly once starting and ending at the same vertex (see Figure 13).

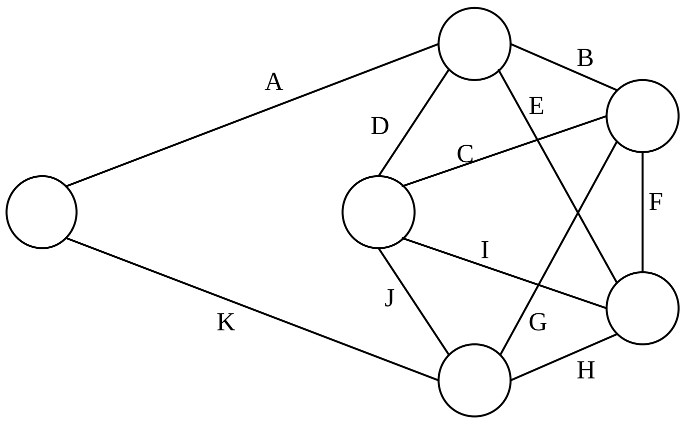

Figure 13: A Eulerian graph

7.4 Lagrangean Relaxation[64]

The Lagrangean Relaxation bases on the idea that many hard problems can be seen as easy problems which are too complex because of a relatively small number of constraints. If these are dualized, an easily solved Lagrangean problem is generated. Its optimal solution can now be used for a lower bound in the original problem in a branch and bound algorithm.

[64] Fisher, 2004, p. 1861

In 1970 Held and Karp used a Lagrangean relaxation for solving an algorithm for the traveling salesman problem which has been extremely successful. This constitutes the “birth” of the Lagrangean technique.

The concept behind this approach is that hard capacity constraints are moved into the objective function and penalty costs, the so called Lagrangean multipliers, are imposed on their violation. This way the problem can be solved rather easily.

7.5 Monge Array[65]

A Monge array is an m x n array for which the following property holds:

$$A[i,\ j]+A[k,\ l]\leq A[i,\ l]+A[k,\ j] \qquad\qquad \forall\ i,\ \ j,\ k,\ l$$

if $1\leq i<k\leq m$ and $1\leq j<l\leq m$. One can also say that if we choose two rows and two columns of the array the sum of the upper-left and lower-right intersecting elements will be less or equal to the sum of the lower-left and upper-right intersecting elements. Table 3 shows an example of a Monge array.

10	**17**	13	28	**23**
17	**22**	16	29	**23**
24	**28**	**22**	**34**	**24**
11	**13**	6	17	**7**
45	**44**	**31**	**37**	**23**
36	**33**	19	21	**6**
75	**66**	51	53	**34**

Table 3: A Monge array[66]

If we take the rows 3 and 5 and the columns 2 and 5 then we can see that the first sum $28+23=51$ is smaller than the second sum $44+24=68$. The same property holds if we make another choice.

[65] Cormen et al., 2001, p. 88

[66] Cormen et al., 2001, p. 88

8 Summary

In this diploma thesis the Traveling Repairman Problem (TRP) has been examined. First an overview of the Traveling Salesman Problem (TSP) was given, of which the TRP is a variation. Then the characteristics and the mathematical formulation of the TRP have been described. In the next chapter some solving methods for the TRP have been introduced to get a better understanding for the problem, a branch and bound algorithm which leads to an optimal solution, an exact algorithm and a heuristic both based on dynamic programming.

Afterwards some variations of the TRP with a single repairman were presented, of which the line-TRP was the first. It is a variant of the TRP with a line as the underlying graph and the versions with and without deadlines were examined, meaning that the customers have deadlines at which they must have been served. Then the line-TRP with weights has been introduced, where a weight is assigned to each customer. This ensures that some customers, which are more important, are served earlier than others. In the next subchapter the variant of the TRP with a directed underlying graph was presented. Here the distances between two vertices depend on the direction the repairman travels. Then the TRP on weighted trees has been examined. Here the weights on the edges of the tree represent the distances between any two vertices. Afterwards the weighted TRP on a metric space has been introduced, where once again the weights assigned to the vertices show the importance of the corresponding customers. The last model described in this chapter was the on-line TRP where the orders of the customers are not known in advance but come in over time.

The next chapter examined variants of the TRP with more than one repairman available to serve the orders of the customers (the k-TRP). First a general problem has been described, then the k-TRP with repairtimes was introduced, where a repairtime is assigned to each customer which can differ from one to another. Finally the on-line k-TRP was presented.

Afterwards some applications for the TRP have been described of which two have been examined in more detail. The first was an emergency vehicle dispatching system which has been established for an electric utility company in Chile and the second described the technician and task scheduling probem (TTSP) which also uses parts of the TRP.

In the last chapter some basis, for example graph theory or the definition of a Monge Array, and solving methods as the branch and bound technique or the lagrangean relaxation have been presented.

The TRP is very challenging and is even currently examined frequently because of its many application possibilities. Additionally it is NP-hard which also makes it to an issue of general interest in the literature. For these problems it is very difficult to find an optimal solution in a reasonable amount of time. Furthermore one can expand the model by adding constraints depending on the part of the problem on which one concentrates or depending on the problem one tries to solve. By combining them it is possible to create a model which exactly suits the desired characteristics of the problem that has to be solved.

IV. References

1. Literature in alphabetical order:

- Afrati, F., Cosmadakis, S., Papadimitriou, C. H., Papageorgiou, G., Papakostantinou, N., *The Complexity of the Traveling Repairman Problem*, Rairo – Theoretical Informatics and Applications, Vol. 20, No. 1, 1986, pp. 79-87
- Allulli, Luca, Ausiello, Giorgio, Bonifaci, Vincenzo, Laura, Luigi, *On the Power of Lookahead in On-Line Server Routing Problems*, Theoretical Computer Science, Vol. 408, No. 2-3, 2008, pp. 116-128
- Bianco, Lucio, Mingozzi, Aristide, Ricciardelli, Salvatore, *The Traveling Salesman Problem with Cumulative Costs*, Networks, Vol. 23, No. 2, 1993, pp. 81-91
- Blum, Avrim, Chalasani, Prasad, Coppersmith, Don, Pulleyblank, Bill, Raghavan, Prabhakar, Sudan, Madhu, *The Minimum Latency Problem*, Proceedings of the 26th ACM Symposium on the Theory of Computing, 1994, pp. 163-171
- Bonifaci, Vincenzo, Stougie, Leen, *Online k-Server Routing Problems*, Lecture Notes in Computer Science, WAOA 2006, Vol. 4368, 2006, pp. 83-94
- Chaudhuri, Kamalika, Godfrey, Brighten, Rao, Satish, Talwar, Kunal, *Paths, Trees, and Minimum Latency Tours*, Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science, 2003, pp. 36-45
- Cordeau, Jean-François, Laporte, Gilbert, Pasin, Frederico, Ropke, Stefan, *Scheduling Technicians and Tasks in a Telecommunications Company*, 2008, pp. 1-25
- Cormen, Thomas H., Leiserson, Charles E., Rivest, Ronald L., Stein, Clifford, *Introduction to Algorithms*, Second Edition, MIT Press, 2001
- Fakcharoenphol, Jittat, Harrelson, Chris, Rao, Satish, *The k-Traveling Repairmen Problem*, ACM Transactions on Algorithms, Vol. 3, No. 4, 2007, Article 40, pp. 40:1-40:16
- Fischetti, Matteo, Laporte, Gilbert, Martello, Silvano, *The Delivery Man Problem and Cumulative Matroids*, Operations Research, Vol. 41, No. 6, 1993, pp. 1055-1064

- Fisher, Marshall L., *The Lagrangian Relaxation Method for Solving Integer Programming Problems*, Management Science, Vol. 50, No. 12, 2004, pp. 1861-1871
- García, Alfredo, Jodrá, Pedro, Tejel, Javier, *A Note on the Travelling Repairman Problem*, Networks, Vol. 40, No. 1, 2002, pp. 27-31
- García, Alfredo, Jodrá, Pedro, Tejel, Javier, *An Efficient Algorithm for On-Line Searching of Minima in Monge Path-Decomposable Tridimensional Arrays*, Information Processing Letters, Vol. 68, 1998, pp. 3-9
- Goemans, Michel, Kleinberg, Jon, *An Improved Approximation Ratio for the Minimum Latency Problem*, Mathematical Programming, Vol. 82, No. 1-2, 1998, pp. 111-124
- Jothi, Raja, Raghavachari, Balaji, *Approximating the k-Traveling Repairmen Problem with Repairtimes*, Journal of Discrete Algorithms, Vol. 5, No. 2, 2007, pp. 293-303
- Jothi, Raja, Raghavachari, Balaji, *Minimum Latency Tours and the k-Traveling Repairmen Problem*, Lecture Notes in Computer Science, LATIN 2004, Vol. 2976, 2004, pp. 423-433
- Krumke, S. O., de Paepe, W. E., Poensgen, D., Stougie, L., *News from the Online Traveling Repairman*, Theoretical Computer Science, Vol. 295, No. 1-3, 2003, pp. 279-294
- Lawler, E. L., Lenstra, J. K., Rinnooy Kan, A. H. G., Shmoys, D. B., *The Traveling Salesman Problem*, John Wiley & Sons Ltd., Great Britain, 1985
- Nagarajan, Viswanath, Ravi, R., *The Directed Minimum Latency Problem*, Lecture Notes in Computer Science, Vol. 5171, 2008, pp. 193-206
- Sarubbi, J. F. M., Luna, H. P. L., *A New Flow Formulation for the Minimum Latency Problem*, In: International Network Optimization Conference (INOC), Spa, Belgium, 2007
- Weintraub, A., Aboud, J., Fernandez, C., Laporte, G., Ramirez, E., *An Emergency Vehicle Dispatching System for an Electric Utility in Chile*, The Journal of the Operational Research Society, Vol. 50, No. 7, 1999, pp. 690-696

- Wu, Bang Ye, *Polynomial Time Algorithms for Some Minimum Latency Problems*, Information Processing Letters, Vol. 75, No. 5, 2000, pp. 225-229
- Wu, Bang Ye, Huang, Zheng-Nan, Zhan, Fu-Jie, *Exact Algorithms for the Minimum Latency Problem*, Information Processing Letters, Vol. 92, No. 6, 2004, pp. 303-309

2. Internet:
   - http://en.wikipedia.org/wiki/Branch_and_bound; 05.06.2009
   - http://en.wikipedia.org/wiki/Graph_(mathematics)#Undirected_graph; 05.06.2009
   - http://en.wikipedia.org/wiki/Dynamic_programming; 18.08.2009

Abstract

Diese Magisterarbeit gibt einen Überblick über das Traveling Repairman Problem (TRP), das eine Spezialform des Problems des Handlungsreisenden (Traveling Salesman Problem – TSP) darstellt. Beide Modelle werden benutzt, um die Tour eines Handlungsreisenden zu planen, der in einer vorgegebenen Zeitspanne eine bestimmte Anzahl von Kunden besuchen soll. Während das TSP sich darauf konzentriert, die Länge der Tour zu minimieren, versucht das TRP, die Summe der Wartezeiten der Kunden so gering wie möglich zu halten.

Der Hauptteil der Arbeit beschäftigt sich mit der Definition und den Varianten des TRP und beschreibt mögliche Modelle und Verfahren, mit deren Hilfe diese zu lösen sind. Dabei werden zuerst die Problemstellungen definiert und dann die mathematischen Formulierungen bzw. die Algorithmen dargestellt.

Zu Beginn der Arbeit werden das TSP und das TRP näher definiert und kurz anhand eines Beispiels illustriert (in Kapitel 2). Danach werden das allgemeine TRP und einige Lösungsverfahren dazu näher erläutert (in Kapitel 3).

Im Hauptteil werden zuerst einige Variationen des TRP mit einem einzelnen Repairman und Algorithmen zur Lösung dieser Modelle beschrieben (in Kapitel 4). Dann werden das TRP mit mehreren Repairmen sowie einige Spezialformen hierzu erläutert (in Kapitel 5).

Zusätzlich werden in dieser Arbeit Anwendungsmöglichkeiten beschrieben, von denen zwei genauer untersucht werden (in Kapitel 6). Schließlich werden noch einige Basisbegriffe und Lösungsmethoden erläutert (in Kapitel 7).

Curriculum vitae

Name:	Miriam Lechmann
Geboren:	16.3.1978, in Wien
Staatsbürgerschaft:	Österreich
Familienstand:	ledig

Schulbildung:

1984-1988	Volksschule, Wien
1988-1996	Neusprachliches Gymnasium, Wien 1996 Matura (mit gutem Erfolg abgeschlossen)

Studium:

1997-2009	Diplomstudium IBWL Universität Wien (BWZ)
KFKs:	KFK Produktionsmanagement am Lehrstuhl für Produktion und Logistik KFK Kapitalmarktforschung und Investmentanalyse am Lehrstuhl für Finanzwirtschaft und Banken

Beruflicher Werdegang:

1997–2001	AXA Nordstern Colonia Versicherung, Wien geringfügige Tätigkeit im Sekretariat der Generaldirektion
2001–2004	Verlag Carl Ueberreuter, Wien geringfügige Tätigkeit in der Rechte- und Lizenzabteilung
2004–2006	Verlag Carl Ueberreuter, Wien Lektoratsassistentin
2006–2008	OSRAM, Wien Controlling-Mitarbeiterin und Assistentin des Compliance-Officers
2009	Verlag Carl Ueberreuter, Wien Assistentin der Geschäftsführung