university of groningen / faculty of mathematics and natural sciences

# Traveling Salesman Problem

# Comparisons between heuristics, linear and semidefinite programming approximations

Mariëlle Kruithof

Master's Thesis in Applied Mathematics

29 June 2012

# Traveling Salesman Problem

# Comparisons between heuristics, linear and semidefinite programming approximations

**Summary**

In this thesis we investigate linear and semidefinite programming approximations for the Traveling Salesman Problem: the problem of finding the tour of minimum length that visits each city exactly ones. We conduct numerical comparisons between the Held-Karp and the Van der Veen relaxations for the symmetric circulant Traveling Salesman Problem. Out of these comparisons we conjecture that these bounds are equal. Furthermore, we construct a heuristic to find a tour out of the solution matrix of the best existing semidefinite programming approximation of the Traveling Salesman Problem. Most of the time our method gives a tour value closer to the optimum than existing tour construction methods like nearest-neighbor and farthest insertion.

Master's Thesis in Applied Mathematics
Author: Mariëlle Kruithof
First supervisor: Dr. Cristian Dobre
Second supervisor: Prof. Dr. Mirjam Dür
Date: 29 June 2012

Johann Bernoulli Institute for Mathematics and Computer Science
P.O. Box 407
9700 AK Groningen
The Netherlands

# Contents


**1 Preliminaries** **1**
- 1.1 General information about the TSP . . . 1
- 1.2 Introductory part on semidefinite programming . . . 2
  - 1.2.1 Some history . . . 2
  - 1.2.2 Structure . . . 3
- 1.3 Using linear programming approximations for the TSP . . . 4
- 1.4 Using semidefinite programming for the TSP . . . 7
- 1.5 Particular cases of the TSP . . . 8
  - 1.5.1 Metric Traveling Salesman Problem (MTSP) . . . 8
  - 1.5.2 Euclidean Traveling Salesman Problem (ETSP) . . . 8
  - 1.5.3 Asymmetric Traveling Salesman Problem (ATSP) . . . 8
  - 1.5.4 Circulant Traveling Salesman Problem (CTSP) . . . 9
- 1.6 Heuristics to solve the TSP . . . 11
  - 1.6.1 Tour construction methods . . . 11
  - 1.6.2 Tour improvement methods . . . 12
  - 1.6.3 Comparing bounds . . . 13

**2 Circulant TSP** **15**
- 2.1 Introduction . . . 15
  - 2.1.1 Polytopes and polyhedra . . . 15
- 2.2 Held-Karp bound . . . 16
  - 2.2.1 Generating subtour elimination constraints . . . 17
- 2.3 Van der Veen bound . . . 21
- 2.4 Numerical comparisons . . . 22
  - 2.4.1 Numerical results . . . 24

**3 Heuristics and tours** **27**
- 3.1 Heuristics . . . 27
  - 3.1.1 Nearest-neighbor . . . 27
  - 3.1.2 Farthest insertion . . . 28
  - 3.1.3 Biased random sampling . . . 29
  - 3.1.4 Euclidean symmetric TSP . . . 31
- 3.2 Using linear and semidefinite programming for TSP . . . 34
  - 3.2.1 Solution matrix $X$ . . . 35
  - 3.2.2 Constructing a tour . . . 37
  - 3.2.3 Comparing the constructed tour value with the heuristics . . . 39

3.2.4 Conclusion . . . . . 42

**A Heuristics** **43**
A.1 Adjacency . . . . . 43
A.2 Nearest-neighbor . . . . . 43
A.3 Farthest insertion . . . . . 44
A.4 Biased random sampling . . . . . 46

**B Van der Veen** **49**

**C Held-Karp** **51**
C.1 Maximum flow . . . . . 51
C.2 Held-Karp bound . . . . . 52

**D Semidefinite programming** **55**

**E Tour construction** **57**

**Bibliography** **59**

# Chapter 1

# Preliminaries

The aim of this chapter is to present the knowledge necessary for the research conducted in this thesis.
In the first section we will give some general background of the Traveling Salesman Problem (TSP). Once we are familiar with this concept we will introduce the semidefinite programming problem (SDP); and use it to approximate the optimal value of the TSP. In the second section we will present the general structure of SDP after presenting some history. Semidefinite programming is a generalization of linear programming (LP), therefore in the third section we will first give an LP approximation for the Traveling Salesman Problem to come up with an SDP approximation for the TSP in the next section.
This thesis will also bring in discussion a special form of the Traveling Salesman Problem: the symmetric circulant TSP. Therefore the fifth section provides several special forms of the TSP, with the emphasis on the circulant TSP. We will end this chapter by presenting heuristics to solve the TSP.

## 1.1 General information about the TSP

Although today a lot of our correspondence occurs electronically, still every day hundreds of thousands of postals are sent by 'normal' mail. To get the sent postals at their destination, tens of thousands of postmen deliver the postals. In order to make sure that every postman can deliver as much postals as possible, it is necessary to have a good schedule of the route they will have to follow. A good schedule can be achieved in multiple ways. For example if we want to obtain the total traveled distance as small as possible we write down the distances between all the addresses and try to find the best order of addresses the postman has to follow. This order gives a route of minimal total length. The shortest route, however, is not always the fastest. It may be that this route costs a lot of time to travel. So instead of minimizing the traveled distance, we can also minimize the total traveled time.
The problem of finding the optimal route (i.e. the optimal tour), is called the Traveling Salesman Problem (TSP). The optimal tour can thus be found using different objectives like time, distance or weight. For mathematical purposes all these variants are equivalent. The models in this thesis will apply to any of the considerations.
By assuming that the way back and forth between two addresses is the same, we can represent the TSP by considering an undirected graph $G = (V, E)$, where $V$ denotes the number of vertices and $E$ denotes the number of edges of the graph. Here the addresses the postman

has to visit are represented by the vertices and the connections between the addresses are represented by the edges. The distance between the addresses is not equal, therefore we also define a distance matrix $D$ with $D_{ij} > 0$ if $i \neq j$ and 0 otherwise.
A route visiting each vertex of the graph $G$ exactly once and returning to the starting vertex is called a Hamiltonian cycle. If we can find the route between the vertices with the lowest total distance by visiting each vertex exactly once, we have found the optimal solution to the TSP.
Here we assumed that $D_{ij} > 0$, but the most general form of the distance matrix can contain any value: positive, negative as well as infinity. Infinity will be set if there does not exist an edge between two vertices. If there are negative entries in the matrix, adding the value of at least the smallest entry to every entry in the matrix will result in a matrix with only positive entries. Since we add the same value to every entry it does not influence the minimal tour. Therefore in this thesis we will assume that every distance matrix contains only positive entries.

**Definition 1.1.1.** *The Traveling Salesman Problem is the problem of finding the Hamiltonian cycle in $G$ with minimum length.*

The TSP is a combinatorial optimization problem. These are optimization problems, so that for any instance the solution set is finite or countable. The optimal solution to the TSP can be found using the algorithm: try all possible routes. However this is very time-consuming work. In a graph with $n$ vertices there are $n!$ different routes. For low $n$ the optimal solution can be found trying all the possible routes, for higher $n$ it becomes impossible. For example if we could evaluate one million tours per second with a fast computer, a 15-city problem already takes 1.5 hour computing time, a 20-city problem even more than 750 centuries [24]. This is far too long to wait for!
Therefore we need the use of algorithms that can solve or approximate the problem in faster (i.e. polynomial) time.
Algorithms that solve the problem in polynomial time belong to the class $P$. If we can only find nondeterministic polynomial time algorithms to solve the problem, the problem belongs to the class $NP$. The general TSP is part of the $NP$-hard problems [2]. $NP$-hard problems cannot be solved in polynomial time to optimality, unless $P = NP$.
To be able to obtain a solution for the TSP in reasonable time, we will make use of approximations that give a solution to the problem in polynomial time. In this thesis we will encounter linear (polyhedral) and semidefinite programming (nonpolyhedral) relaxations.

## 1.2 Introductory part on semidefinite programming

### 1.2.1 Some history

An approach to solve combinatorial optimization problems like the Traveling Salesman Problem is to form the polytope associated to the set of feasible solutions ($P$) and then use linear programming to optimize over $P$. See Section 2.1.1 for details on polytopes. However, only partial description of the Traveling Salesman polytope is known, therefore we obtain relaxations. The start of linear programming is usually attributed to George B. Dantzig and John von Neumann, who respectively established the simplex method to solve LP and the theory of duality in 1947 [18]. Semidefinite programming is an extension of linear programming

by using matrix variables, however the simplex method cannot be extended to semidefinite programming.
One way to solve SDP problems to any fixed accuracy in polynomial time is by using the ellipsoid algorithm of Nemirovski and Yudin, developed in the 1970s [13].
In 1984 Karmarkar established a new method to solve linear programming problems: the interior point method. In 1988 Nesterov and Nemirovski, and independently from them Alizadeh and Kamath and Karmarkar showed that the interior point methods could be extended to semidefinite programming. When it appeared to be hard to exploit general sparsity in SDP problem data, new interest was born to find structures in SDP data that allow problem size reduction [23]. In this framework the work of De Klerk, Pasechnik and Sotirov, see [15], provided a new relaxation for the TSP.

### 1.2.2 Structure

The optimal solution of a semidefinite programming problem provides the optimal location and the optimal value of the objective. Therefore, in standard form, we want to find a matrix $X = (X_{ij}) \in \mathbb{R}^{n\times n}$ that solves the problem given matrices $A^{(0)}, \ldots, A^{(m)} \in \mathbb{R}^{n\times n}$ and given a vector $b \in \mathbb{R}^m$. Recall we denoted the vector space of real symmetric matrices by $\mathbb{S}^{n\times n}$.
If $A^{(k)}_{ij}$ denotes the $ij$th entry of the matrix $A^{(k)}$ with $k = 0, \ldots, m$, then the standard SDP problem consists of three parts:

1. *Objective*: minimize or maximize $\sum_{i,j=1}^{n} A^{(0)}_{ij} X_{ij}$;

2. *Linear equality constraints*: $\sum_{i,j=1}^{n} A^{(k)}_{ij} X_{ij} = b_k$, for $k = 1, \ldots, m$;

3. *Positive semidefinite constraint*: $X = (X_{ij}) \in \mathbb{S}^{n\times n}_{+}$,

where the last constraint means that $X$ will be symmetric positive semidefinite, also denoted by $X \succeq 0$.
As can be seen in Section 7.2 in [9], the following five equivalence relations hold for a symmetric matrix $X$:

1. $X \succeq 0$;

2. $z^T X z \geq 0 \ \forall z \in \mathbb{R}^n$;

3. All eigenvalues of $X$ are nonnegative;

4. The determinants of all the principal minors of $X$ are nonnegative;

5. $X = LL^T$ for some $L \in \mathbb{R}^{n\times n}$.

This last equivalence is also called the Cholesky decomposition of matrix $X$.
The trace operator can be used to write

$$\sum_{i,j=1}^{n} A_{ij} X_{ij} = \text{trace}(AX). \tag{1.1}$$

The semidefinite programming problem can thus be written as follows:

$$\begin{array}{ll} \min/\max & \operatorname{trace}(A_0 X) \\ \text{subject to} & \operatorname{trace}(A_i X) = b_i, \quad i = 1, \ldots, m \\ & X \succeq 0. \end{array} \tag{1.2}$$

We write $X \succ 0$ when we have a nonsingular $X \succeq 0$. This means that our matrix $X$ is positive definite. The corresponding set is denoted by $\mathbb{S}^{n\times n}_{++}$.

## 1.3 Using linear programming approximations for the TSP

This thesis focuses mainly on semidefinite programming approximations for the TSP. We also present comparisons with LP approximations, therefore this section introduces LP programs used to approximate the TSP.
Given an undirected graph $G = (V, E)$, $|V| = n$ and the edge lengths $c_e \in E$, we can approximate the value of an optimal tour $T$ for the TSP if we consider the following LP relaxation, known as the Held-Karp bound [7], [25]:

$$\begin{aligned} HK := \min & \sum_{e\in E} c_e x_e \\ \text{s.t.} & \sum_{e\in\delta(v)} x_e = 2, && \forall\, v \in V && (1) \\ & \sum_{e\in S} x_e \leq |S| - 1, && \forall\, \emptyset \subset S \subset V && (2) \\ & x_e = \begin{cases} 1, \text{ if } x_e \in T \\ 0, \text{ otherwise,} \end{cases} \end{aligned} \tag{1.3}$$

where the cut $\delta(v)$ denotes the number of edges leaving the vertex $v$. The first constraint (1) describes that the number of all edges incident to a vertex $v$ in a tour is equal to 2. The second constraint (2) shows that if we have a non-empty subset $S$ of $V$ the total number of edges leaving $S$ is less than or equal to the number of nodes of $S$ minus 1. These are called the subtour elimination inequalities. We have exponentially many subtour elimination inequalities $(2^n - 2) = \binom{n}{1} + \binom{n}{2} + \cdots + \binom{n}{n-1}$. However using the ellipsoid method the problem can be solved in polynomial time as proven by Schrijver in [21].
By rewriting the first and second constraint we obtain the following LP:

$$\begin{aligned} HK := \min & \sum_{e\in E} c_e x_e \\ \text{s.t.} & \sum_{e\in E} x_e = n, && && (3) \\ & \sum_{e\in\delta(S)} x_e \geq 2, && \forall\, \emptyset \subset S \subset V && (4) \\ & x_e = \begin{cases} 1, \text{ if } x_e \in T \\ 0, \text{ otherwise,} \end{cases} \end{aligned} \tag{1.4}$$

where $\delta(S)$ denotes the cut between $S$ and $V\backslash S$. Now the first constraint (3) means that the number of edges in the tour $T$ is equal to the number of vertices and the second constraint (4) is bounding from below the number of edges connecting $S$ with $V\backslash S$ to a minimum of 2 for all $S$. This is again the subtour inequality constraint.
We will prove that (1.3) and (1.4) are equivalent.

**(1.3) $\Rightarrow$ (1.4)**

We will derive (3) out of (1) in the following way:

$$\sum_{e\in\delta(v)} x_e = 2.$$

If we take the sum over all $n$ vertices, this yields:

$$\sum_{v\in V}\sum_{e\in\delta(v)} x_e = 2n.$$

Each edge is connected to two vertices, giving:

$$\sum_{v\in V}\sum_{e\in\delta(v)} x_e = 2\sum_{e\in E} x_e,$$

therefore:

$$\sum_{e\in E} x_e = n,$$

which is (3).
We use this result, together with inequality (2) to derive (4).
Given (3):

$$\sum_{e\in E} x_e = n = \sum_{e\in S} x_e + \sum_{e\in V\backslash S} x_e + \sum_{e\in\delta(S)} x_e.$$

And given (4):

$$\sum_{e\in S} x_e \leq |S| - 1,$$

which implies:

$$\sum_{e\in V\backslash S} x_e \leq |V\backslash S| - 1.$$

Together this yields:

$$\sum_{e\in S} x_e + \sum_{e\in V\backslash S} x_e + \sum_{e\in\delta(S)} x_e \leq |S| - 1 + |V\backslash S| - 1 + \sum_{e\in\delta(S)} x_e.$$

Since $|S| + |V\backslash S| = |V| = n$ we now have:

$$n \leq n - 2 + \sum_{e\in\delta(S)} x_e.$$

Hence

$$\sum_{e\in\delta(S)} x_e \geq 2,$$

which is (4).

**(1.4) ⇒ (1.3)**

We will derive (1) out of (3) and (4):
We know (4)

$$\sum_{e\in\delta(S)} x_e \geq 2 \quad \forall\, \emptyset \subset S \subset V.$$

Take $S = \{v\} \subset V$, then

$$\sum_{e\in\delta(v)} x_e \geq 2 \quad \forall v \in V. \quad \text{(4b)}$$

Every edge is connected to two vertices, so we take twice constraint (3) resulting in:

$$2n = 2\sum_{e\in E} x_e = \sum_{v\in V}\sum_{e\in\delta(v)} x_e.$$

Together with (4b) this yields

$$\sum_{e\in\delta(v)} x_e = 2 \quad \forall v \in V,$$

which is (1).
This result we use, together with inequality (4) to derive (2):
Given (1):

$$\sum_{e\in\delta(v)} x_e = 2 \;\forall v \in V,$$

if we take the sum over all vertices of the subset $S$ this implies:

$$\sum_{v\in S}\sum_{e\in\delta(v)} x_e = 2\,|S| \quad \forall\, \emptyset \subset S \subset V.$$

The edges counted in $\sum_{v\in S}\sum_{e\in\delta(v)} x_e$ are the edges of the cut of $S$ and twice the edges in $S$:

$$\sum_{v\in S}\sum_{e\in\delta(v)} x_e = \sum_{e\in\delta(S)} x_e + 2\sum_{e\in S} x_e,$$

hence

$$\sum_{e\in\delta(S)} x_e + 2\sum_{e\in S} x_e = 2\,|S|\,.$$

Together with constraint (4) $\sum_{e\in\delta(S)} x_e \geq 2$, this implies:

$$2\sum_{e\in S} x_e \leq 2\,|S| - 2,$$

hence

$$\sum_{e\in S} x_e \leq |S| - 1,$$

which is (2).

In this way we proved that (1.3) and (1.4) are equivalent.

The Held-Karp bound can also be written in matrix form:

$$\begin{aligned} HK := \min\ & \frac{1}{2}\text{trace}(DX) \\ \text{s.t. } & Xe = 2e, \\ & \text{diag}(X) = 0, \\ & 0 \le X \le J, \\ & \sum_{i\in I, j\notin I} X_{ij} \ge 2 \quad \forall\ \emptyset \neq I \subset \{1,\dots,n\}, \end{aligned} \tag{1.5}$$

where $D$ denotes the distance matrix, $e$ the all-ones vector and $J$ the all-ones matrix. The inequalities $0 \le X \le J$ are component wise. Notice that all the constraints are linear, so we still have an LP formulation.
The first constraint makes sure we get a tour: every vertex has two edges leaving from it. This is related to (1) of (1.3). Since we do not want a connection of a vertex with itself (i.e. loops), we will set the diagonal of $X$ to zero. We want the maximum of edges between two vertices to be one, therefore we state that the matrix $X$ contains only entries between 0 and 1. The last constraint is the subtour elimination constraint, related to (4) of (1.4). For every nonempty subset $I$, the number of edges connecting the vertices of $I$ to the vertices that are not in $I$ has a minimum of 2. In this way there cannot be a subset of vertices with a closed tour, hence this constraint eliminates the subtours.

## 1.4 Using semidefinite programming for the TSP

The value of an optimal tour in the Traveling Salesman Problem can also be approximated using semidefinite programming.
Consider the following SDP relaxation [15]:

$$\begin{aligned} KPS := \min\ & \frac{1}{2}\text{trace}(DX^{(1)}) \\ \text{s.t. } & X^{(k)} \ge 0, \quad k = 1,\dots,d \\ & \sum_{k=1}^{d} X^{(k)} = J - I, \\ & I + \sum_{k=1}^{d} \cos(\frac{2ki\pi}{n}) X^{(k)} \succeq 0, \quad i = 1,\dots,d \\ & X^{(k)} \in \mathbb{S}^{n\times n}, \quad k = 1,\dots,d, \end{aligned} \tag{1.6}$$

where $D$ is the distance matrix, $J$ is the $n \times n$ matrix consisting of all ones and $d = \lfloor \frac{n}{2} \rfloor$. Notice that this in indeed an SDP, since the third constraint gives us that $I + \sum_{k=1}^{d} \cos(\frac{2ki\pi}{n}) X^{(k)}$ has to be positive semidefinite for $i = 1,\dots,d$. This is in contrast to the Held-Karp bound in (1.5) where all constraints are linear. De Klerk, Pasechnik and Sotirov showed in [15] that this SDP provides a lower bound on the length of an optimal tour. Further they showed that

for general TSP the Held-Karp bound does not dominate the SDP bound (1.6) or vice versa. This is the best existing SDP bound for the TSP known for the authors.

## 1.5 Particular cases of the TSP

The Traveling Salesman Problem can be found in practice, occurring in different cases. We will mention four particular cases of the TSP: the metric TSP, Euclidean TSP, asymmetric TSP and the circulant TSP.
The metric, Euclidean and asymetric TSP belong to the class of $NP$-hard problems. The complexity of the circulant TSP is still unknown. Since further in this thesis we will deal with symmetric circulant TSP, in this section our focus will be on the circulant TSP.

### 1.5.1 Metric Traveling Salesman Problem (MTSP)

In the metric TSP the distance matrix $D$ is symmetric and the distances fulfill the triangle inequality:

$$D_{ij} \leq D_{ik} + D_{kj}.$$

An example of the MTSP is the warehouse order picking problem as described in [19].
In a warehouse different items are stored. If an order arrives the required items have to be picked out of the storage. In the storage, the items lie at different aisles. The aisles are parallel to each other with the ends connected by two long corridors. A vehicle with enough capacity to pick up all the needed items at once will drive through the aisles to pick up the required items. The vehicle will enter and leave the storage at the same place. The objective is to minimize the total traveled distance of the vehicle.
Due to the specified metric in the storage, the problem can be formulated as a special case of the metric Traveling Salesman Problem.

### 1.5.2 Euclidean Traveling Salesman Problem (ETSP)

The Euclidean TSP is a special form of the metric TSP. Here the distances are the Euclidean distances. In k-dimensions we have:

$$D_{ij} = \sqrt{\sum_{p=1}^{k} (j_p - i_p)^2},$$

where $i_p$ and $j_p$ denote the vertices in the $p$th-dimension.
In one dimension the distance matrix $D$ consists of the absolute value of the difference between the vertices $i$ and $j$:

$$D_{ij} = \sqrt{(i-j)^2}.$$

The postman problem mentioned in Section 1.1 is an example of the Euclidean Traveling Salesman Problem.

### 1.5.3 Asymmetric Traveling Salesman Problem (ATSP)

The asymmetric TSP can be represented by a directed graph. Here $D_{ij} \neq D_{ji}$. If in our postman example there occurs a one-way between two addresses, the distance to travel between

these addresses is not the same traveling back and forth. This yields an asymmetric matrix $D$. The asymmetric distance matrix can be transformed into a symmetric one. This can be useful when there is only an algorithm available to solve the symmetric TSP. Next to it, if only part of the matrix is asymmetric, we do not have to use the algorithms developed for solving the ATSP. A transformation of the ATSP in the STSP can be found in [11]. Jonker and Volgenant showed that an asymmetric TSP with $n$ nodes is equivalent to a symmetric TSP with at most $2n$ nodes [10].

### 1.5.4 Circulant Traveling Salesman Problem (CTSP)

In the circulant TSP the distance matrix $D$ is circulant.
A circulant matrix is of the following form:

$$D := \begin{pmatrix} d_0 & d_1 & d_2 & . & . & d_{n-1} \\ d_{n-1} & d_0 & d_1 & d_2 & . & d_{n-2} \\ . & d_{n-1} & d_0 & d_1 & d_2 & . \\ . & . & . & . & . & . \\ d_2 & . & . & d_{n-1} & d_0 & d_1 \\ d_1 & d_2 & . & . & d_{n-1} & d_0 \end{pmatrix}. \tag{1.7}$$

This can also be written analytically as

$$D_{ij} := d_{(j-i)} \text{mod } n.$$

A variant given here is the symmetric circulant TSP (SCTSP). In the SCTSP the distance matrix $D$ is symmetric and circulant. Referring to (1.5) and (1.6) Theorem 2 in [14] proves that there exist an optimal solution (i.e. matrix $X$) also symmetric and circulant.
Since circulant matrices play an important role in the research conducted in this thesis, we will give an extended example. An example of the circulant TSP is minimizing wallpaper waste as proposed in [5] and described in [24].

Suppose we want to cover a wall with wallpaper. We have a long roll of wallpaper. The wall has a length of $L$ and a width of $W$. The roll has a width of $w$. This means we need $n = \lceil \frac{W}{w} \rceil$ sheets of wallpaper to cover the whole wall. For imprecise cutting we add a length $\epsilon$ to the length of each sheet. Since we have to cut twice for each sheet, the length of the sheet becomes $L' = L + 2\epsilon$. There is a repeated pattern on the sheets, see Figure 1.1. Therefore the second sheet has to shift with respect to the first sheet. We call $d$, the drop, the distance of this shifting. The height of a pattern, also called rapport, will be denoted by $r$.
We want to minimize the waste of the wallpaper when cutting the sheets. Due to the pattern on the sheets it is not always the best way to cut sheet 2 immediately after sheet 1 of the roll. We want to know in which order the sheets can be cut best of the roll.
The sheets are numbered from 0 to $n-1$. We place the sheets from left to right on the wall. The first sheet, sheet 0, can directly be cut of the roll and will cause zero waste. We want to know the waste if we cut the sheets in any different order of the roll. This information we use to construct a distance matrix.
Let $A_i$ denote the place where the top of sheet $i$ is cut off from the roll, measured as the distance from the top of the pattern. This is exactly the shift with respect to the first sheet, therefore:

$$A_i = (d \cdot i) (\text{mod } r).$$

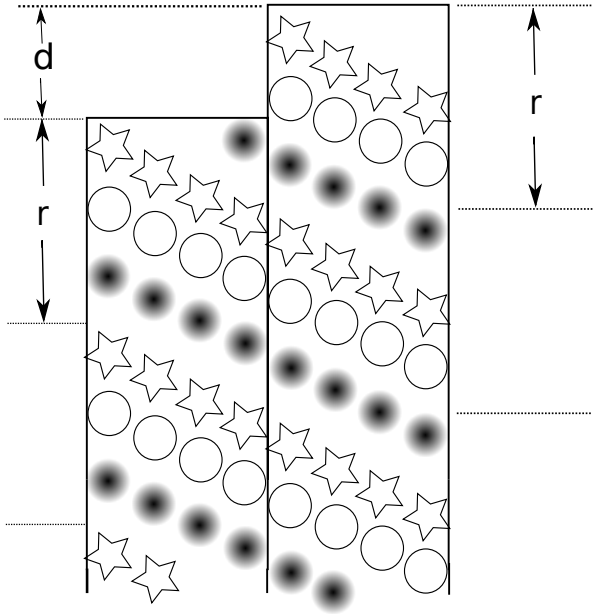


Figure 1.1: Two sheets of wallpaper with drop d and report r

Thus $A_0 = 0$, $A_1 = d \bmod r$, $A_2 = 2d \bmod r$ and $A_{n-1} = (n-1)d \bmod r$. It appears that sheet $m$ with $m = r/\gcd(r,d)$ has the same outcome as sheet 0. As sheet $m$ and sheet 0 have the same outcome, also sheet $m+1$ and sheet 1 have the same outcome. As a consequence we only have to look at the first $m-1$ sheets.

If the top of sheet $i$, for $i = 1, \ldots, m-1$, is at $A_i$, the end of the sheet is $L'$ further. There we have to cut again, so we have to cut at a distance $A_i + L' \bmod r$, measured as the distance from the top of the pattern.

If we cut the end of the first sheet at a distance $A_0 + L' \bmod r$, the start of the second sheet has to be cut at a distance $A_1 \bmod r$. So the waste between the first two sheets is $A_1 - (A_0 + L') \bmod r$. For the entries of the distance matrix we get:

$$\begin{aligned} d_{ij} &:= (A_j - (A_i + L')) \bmod r \\ &= ((j-i)d - L') \bmod r, \end{aligned}$$

for $i = 1, \ldots, m-1$ with $m = r/\gcd(r,d)$.

The distance matrix $D$ that appears is a circulant matrix. These matrices for the wallpaper problem are also called Garfinkel matrices, referring to Robert Garfinkel who first described this problem. This problem can be described as a special case of the circulant TSP.

### A small example

Let $W = 10$, $w = 1.1$, $L' = 15$, $r = 5$ and $d = 2$. This gives $n = \lceil \frac{W}{w} \rceil = \lceil \frac{10}{1.1} \rceil = 10$ and $m = r/\gcd(r,d) = 5/\gcd(5,2) = 5/1 = 5$. So our distance matrix will be $5 \times 5$. The entries of our distance matrix are given by $d_{ij} = ((j-i)d - L') \bmod r$, that is $d_{ij} = ((j-i)2 - 15) \bmod 5$.

This gives:

$$D := \begin{pmatrix} 0 & 2 & 4 & 1 & 3 \\ 3 & 0 & 2 & 4 & 1 \\ 1 & 3 & 0 & 2 & 4 \\ 4 & 1 & 3 & 0 & 2 \\ 2 & 4 & 1 & 3 & 0 \end{pmatrix}.$$

## 1.6 Heuristics to solve the TSP

A heuristic is a method to solve the problem in an acceptable time, useful in practice, but not always resulting in the optimal solution. We will distinguish between heuristics for the TSP. On the one hand we have the tour construction methods and on the other hand the tour improvement methods. Tour construction methods involve building a solution by adding a new vertex on each step. Tour improvement methods improve feasible solutions by performing various exchanges [16], [2].

### 1.6.1 Tour construction methods

We will describe three construction methods: the nearest-neighbor heuristic, the insertion methods and biased random sampling.

#### Nearest-neighbor heuristic

The nearest-neighbor heuristic consists of three steps:

1. Start at an arbitrary vertex.
2. Consider the vertices in the neighborhood and pick the nearest vertex. Repeat this step until all vertices are visited.
3. Return to the starting vertex to obtain a tour.

This algorithm is relatively easy to implement. However there are often vertices included with a high distance to each other. A way of getting a better tour is repeating the algorithm $n$ times, each time with an other vertex as starting point. This has a consequence on the complexity, which becomes $O(n^3)$ instead of $O(n^2)$.

#### Insertion methods

Insertion methods all start with two connected vertices. The next step is to add one by one the remaining vertices. The way of adding these vertices depends on the specific chosen insertion method. There exist insertion methods based on the minimum, like the cheapest and the nearest insertion method.

- Cheapest insertion chooses the vertex that causes the least increase of the existing tour.
- Nearest insertion chooses the vertex that has the total minimal distance to all the vertices of the existing tour.

On the other hand you can find the opposite of these methods: the largest and the farthest insertion methods.

- Largest insertion chooses the vertex that causes the most increase of the existing tour.
- Farthest insertion chooses the vertex that has the total maximal distance to all the vertices of the existing tour.

The nearest as well as the farthest insertion have both a complexity of $O(n^2)$. The cheapest and the largest insertion have a complexity of $O(n^2 \log n)$. Counterintuitive, in practice the largest and the farthest insertion seem to work better than the cheapest and nearest insertion [2], [20].

#### Biased random sampling

Biased random sampling starts with uniform sampling. Each vertex has an equal probability of being selected next. The heuristic starts at an arbitrary vertex. By adding weights to the uninserted vertices of the tour the probability of the next vertex to be chosen is biased. The method of assigning the weights to the vertices influences the amount of biasing.
An example of the biased random sampling heuristic can be found in [12]. Here the probability has been biased so that edges of low cost are selected with higher probability. The complexity of this heuristic is $O(n^2)$.

### 1.6.2 Tour improvement methods

The tour improvement methods start with an already existing solution to the problem. We will describe the $r$-opt/Lin-Kernighan method and Tabu search.

#### $r$-opt/Lin-Kernighan

The $r$-opt algorithm works in two steps:

1. Remove $r$ edges from an existing tour.
2. Reconnect the resulting paths in all possible ways. While the new tour is shorter than the original tour, repeat these steps.

The easiest $r$-opt algorithm is the 2-opt. In practice the $r$-opt algorithm with $r > 3$ is seldom used, since the complexity of the algorithm is $O(n^r)$, see [8].
The algorithm is improved by Lin and Kernighan in 1973, who gave their names to the algorithm. The Lin-Kernighan algorithm differs in two ways from the $r$-opt algorithm: the value of $r$ is allowed to vary and when an improvement is found it is not always used immediately [2]. For a detailed description of the algorithm we refer to [17].

#### Tabu search

Tabu search works with a search procedure to move from a solution $x$ to a solution $x'$ in the neighborhood of $x$. In the TSP $x$ is the order in which the vertices are visited. The neighborhood of $x$ can for example consist of the possible solutions obtained by pairwise interchange of the vertices of the solution $x$. Already visited solutions are declared tabu and are placed in a tabu list.
Tabu search can be summarized in three steps:

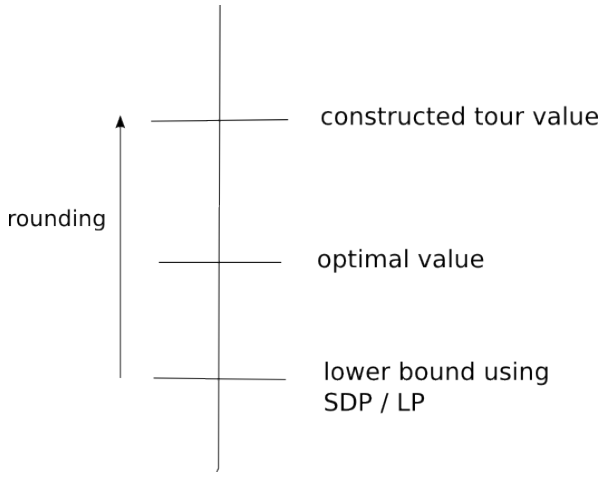


Figure 1.2: From SDP or LP bound to a constructed tour value via rounding.

1. Look at all the neighbors of the current solution, except the ones that are on the tabu list, to find a best-quality solution.
2. Take the best-quality solution as the new tour and update the tabu search.
3. Repeat these steps until the stop criterion has been reached. This stop criterion is often a maximum number of iterations or the maximum running time.

Further details can be found in [6].

### 1.6.3 Comparing bounds

The value of the tour, approximated using linear programming and semidefinite programming is a lower bound to the value of the optimal tour $T$. Since LP and SDP weakened the constraints to obtain a solution to the problem, the entries of our solution matrix $X$ of (1.5) will not consist of only zeros and ones. Instead, the values of the matrix $X$ will lie between zero and one. In the last chapter we will present a heuristic to round the values of $X$ in order to obtain an existing tour. Out of this tour we can derive the constructed tour value. The value of this constructed tour is higher than or equal to the optimal value. Notice that if the value would be less than the value of the optimal tour, we would have found the optimal tour, which contradicts the optimality of $T$.
Further in this thesis we will compare the constructed tour value of the SDP and LP approximation with the value of the tour obtained by some tour construction heuristics of Section 1.6.1, which perform best in practice.

# Chapter 2

# Circulant TSP

In this chapter we consider two linear programming relaxations of the symmetric circulant Traveling Salesman Problem: the Held-Karp relaxation and the Van der Veen relaxation. We conduct a numerical comparison between these relaxations for the symmetric circulant TSP: the problem of finding the Hamiltonian cycle with minimum length in a weighted symmetric circulant graph. Out of the numerical comparison we conjecture that the Held-Karp and the Van der Veen bound are equal.

## 2.1 Introduction

Recall that a circulant distance matrix has the following form:

$$D := \begin{pmatrix} d_0 & d_1 & d_2 & . & . & d_{n-1} \\ d_{n-1} & d_0 & d_1 & d_2 & . & d_{n-2} \\ . & d_{n-1} & d_0 & d_1 & d_2 & . \\ . & . & . & . & . & . \\ d_2 & . & . & d_{n-1} & d_0 & d_1 \\ d_1 & d_2 & . & . & d_{n-1} & d_0 \end{pmatrix}. \tag{2.1}$$

The entries of $D$ can also be written analytically as

$$D_{ij} := d_{(j-i)} \text{mod } n, \quad i, j = 0, \ldots, n-1.$$

We make the symmetry assumption on the distance matrices. This results in matrices of the following form:

$$D := \begin{pmatrix} d_0 & d_1 & d_2 & . & . & d_1 \\ d_1 & d_0 & d_1 & d_2 & . & d_2 \\ . & d_1 & d_0 & d_1 & d_2 & . \\ . & . & . & . & . & . \\ d_2 & . & . & d_1 & d_0 & d_1 \\ d_1 & d_2 & . & . & d_1 & d_0 \end{pmatrix}. \tag{2.2}$$

### 2.1.1 Polytopes and polyhedra

To gain insight in the working of the Held-Karp algorithm we need some information on polytopes and polyhedra. To this end we first define a hyperplane $H$ as

$$H = \{x \in \mathbb{R}^n | c^T x = \delta\},$$

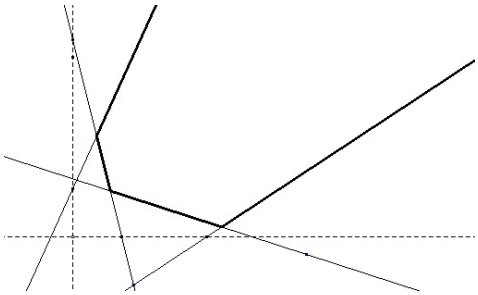

Figure 2.1: Polyhedron: intersection of the halfspaces, area between the thick lines.

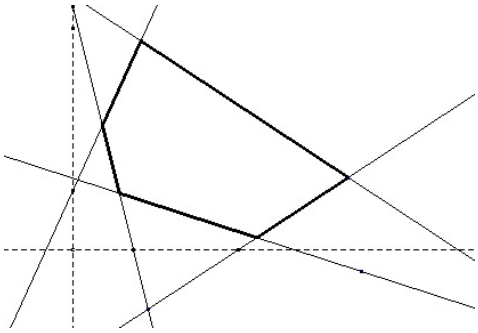

Figure 2.2: Polytope: bounded polyhedron, area between the thick lines.

with $\delta \in \mathbb{R}$ and $c \in \mathbb{R}^n$, with $c \neq 0$.
A hyperplane divides $\mathbb{R}^n$ into two halfspaces. The halfspace $H$ is one of these two parts. For example in $\mathbb{R}^2$ the halfspace is given by the area beneath or above a line drawn through $\mathbb{R}^2$. An affine hyperplane is a hyperplane not necessarily going through the origin.

**Definition 2.1.1.** *A polyhedron is the intersection of a finite number of affine halfspaces.*

An example of a polyhedron in $\mathbb{R}^2$ is shown in Figure 2.1.

**Definition 2.1.1.** *A subset $P$ of $\mathbb{R}^n$ is called a polytope if $P$ is the convex hull of a finite number of vectors.*

An example of a polytope in $\mathbb{R}^2$ is shown in Figure 2.2.

**Theorem 2.1.1.** *A subset $P$ of $\mathbb{R}^n$ is a polytope if and only if it is a bounded polyhedron.*

For a proof of this equivalence we refer to Chapter 2 in [22].

## 2.2 Held-Karp bound

Held and Karp proposed in [7] an LP relaxation for the Traveling Salesman Problem. Solving this relaxation yields a lower bound on the TSP.
Recall that given an undirected graph $G = (V, E)$ and $|V| = n$ the Held-Karp relaxation in

matrix form was introduced in Section 1.3 as follows:

$$\begin{aligned} HK := \min\ & \frac{1}{2}\text{trace}(DX) \\ \text{s.t. } & Xe = 2e, \\ & \text{diag}(X) = 0, \\ & 0 \leq X \leq J, \\ & \sum_{i \in I, j \notin I} X_{ij} \geq 2 \quad \forall\ \emptyset \neq I \subset \{1, \ldots, n\}, \end{aligned} \tag{2.3}$$

where $D$ denotes the distance matrix, $e$ the all-ones vector and $J$ the all-ones matrix. The last constraints are called the subtour elimination constraints. They state that the value of any cut in the corresponding graph $G$ has to be greater than or equal to two. Recall that this means that for any nonempty subset $I$, the number of edges connecting the vertices of $I$ to the vertices that are not in $I$ has a minimum of 2.
The linear programming problem consists of two parts: the objective function $f$ and the constraints. The objective function of the Held-Karp relaxation is the following:

$$f(X) = \min \frac{1}{2}\text{trace}(DX).$$

The constraints can be split into two parts: the subtour elimination constraints and the remaining constraints. We will call this latter set the *body* of the Held-Karp bound. Thus the body includes the following constraints:

$$\begin{aligned} & Xe = 2e, \\ & \text{diag}(X) = 0, \\ & 0 \leq X \leq J. \end{aligned} \tag{2.4}$$

The body is defined by $n^2 + 2n$ linear equalities and inequalities. A matrix $X$ which satisfies the constraints in the body can thus be found in polynomial time.
The last line of (2.3) represents the subtour elimination constraints:

$$\sum_{i \in I, j \notin I} X_{ij} \geq 2 \quad \forall\ \emptyset \neq I \subset \{1, \ldots, n\}. \tag{2.5}$$

The number of the subtour elimination constraints is exponential with a complexity of $O(2^n)$. A matrix $X$ feasible for the relaxation (2.3) can thus not be found in polynomial time. For small $n$ this means that the problem is still solvable in reasonable time, but for larger $n$ the computation time will be hours or even days. In the next section we will describe a way to deal with the subtour elimination constraints without adding to the body all $2^n - 2$ possible inequalities. In that way we can reduce the computation time.

### 2.2.1 Generating subtour elimination constraints

#### Held-Karp bound

The body of the Held-Karp relaxation is a polytope. This polytope consists of all the matrices $X$ that satisfy (2.4). The matrix $X$ gives a representation of a tour. We will abuse terminology and say that the matrix $X$ contains a subtour whenever the corresponding tour does.

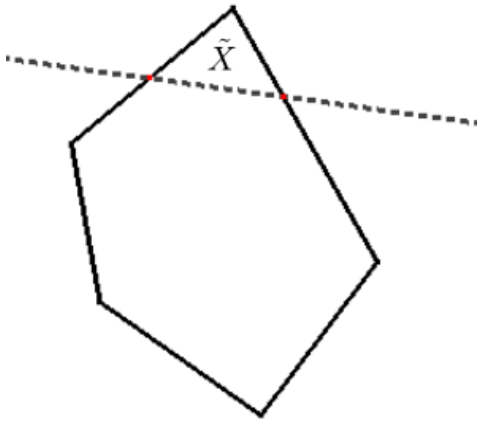


Figure 2.3: The hyperplane cuts a part of the polytope.

The set given by the constraints of the body contains a matrix $\tilde{X}$ such that the value of the objective function $f(\tilde{X})$ is reached. This value $f(\tilde{X})$ represents a lower bound of the TSP ignoring the subtour elimination constraints.

If we include the subtour elimination constraints (2.5) it is possible for some distance matrices that our point $\tilde{X}$ is still valid. This means that adding the subtour elimination constraints to the body does not effect the lower bound $f(\tilde{X})$. However, it is most likely that the solution matrix $\tilde{X}$ does not satisfy the subtour elimination constraints. In this case the solution matrix $X^*$ satisfying (2.4) and (2.5) will result in a higher, thus better, value of the lower bound on the TSP.

In conclusion we get that taking into account the subtour elimination constraints results in a value of the lower bound greater than or equal to the value of the lower bound ignoring the subtour elimination constraints, so $f(X^*) \geq f(\tilde{X})$.

If we take into account the subtour elimination constraints our solution matrix $\tilde{X}$ must satisfy (2.5). This means that we want to have the value of the cut in the corresponding graph $G$ to be greater than or equal to two.

Recall that the cut is the set of edges with endpoints $i \in I$ and $j \notin I$, with $I \subset \{1, \ldots, n\}$. To satisfy (2.5) we check whether this constraint is violated, i.e. we search for the minimum cut in the graph $G$ and see if this value is less than two. If this is the case, this means that our earlier found solution $\tilde{X}$ does not satisfy (2.5) and we reduce our feasible set to exclude $\tilde{X}$ of the set of possible solutions.

Geometrically this means that we draw a hyperplane through the polytope to cut off the part of the polytope containing $\tilde{X}$, see Figure 2.3. In this reduced feasible set we search again for a matrix that satisfies the constraints of the body (2.4). This results in a new solution matrix $\tilde{X}_2$ giving the value $f(\tilde{X}_2)$. For this new solution matrix we check whether the subtour elimination constraints are violated. If these constraints are violated we cut off the part of the polytope containing $\tilde{X}_2$. We continue these steps until there is no more cut with value less than two in the graph. This means that we have found a solution matrix $X^*$ satisfying (2.4) and (2.5), resulting in the Held-Karp bound of the TSP.

### Maximum flow - minimum cut

To be able to find the Held-Karp bound we need to find a matrix $X$ that satisfies all the constraints of the relaxation. For the subtour elimination constraints we want to know the minimum cut. The following theorem gives the relation between the minimum cut and the maximum flow in a graph and is known as the max flow-min cut theorem [1]:

**Theorem 2.2.1.** *The maximum value of a flow in a network equals the minimum capacity of all cuts in this network.*

This theorem tells us that instead of computing the minimum cut we can use the solution of the maximum flow problem in the graph $G$ to find an $X$ that satisfies the subtour elimination constraints.
The maximum flow problem can be stated as follows [1]:

**Definition 2.2.1.** *In the maximum flow problem we wish to send the maximum possible flow in a capacitated network from a specified source vertex $s$ to a specified sink vertex $t$, without exceeding the capacity of any edge.*

Since the flow through an edge between two vertices can not exceed the capacity of that edge, it follows that the value of any flow is less than or equal to the capacity of any cut in the network.
The minimum cut problem can be stated as follows [1]:

**Definition 2.2.1.** *In the minimum cut problem we wish to find the cut with the minimum capacity from among all cuts in the network that separate the source vertex $s$ and the sink vertex $t$.*

This implies that we have found the maximum flow and the minimum cut at the moment that we discover a flow with value equal to the capacity of the cut. This is exactly Theorem 2.2.1. Out of the max flow-min cut theorem we can conclude that a matrix $X$ satisfies constraint (2.5) in the Held-Karp relaxation if there is no maximum flow associated with the network given by $X$ greater than or equal to two.

### Example of max flow-min cut

We illustrate the max flow-min cut theorem in a small example. Suppose we have a directed, capacitated graph with four vertices: a source vertex $s$, vertex 1, vertex 2 and the sink vertex $t$. In Figure 2.4 the vertices, edges and the associated capacities are given. We want to send

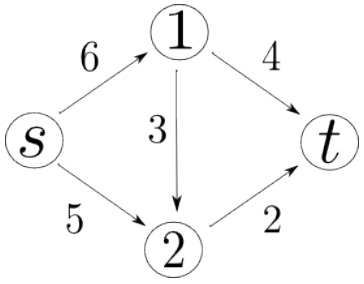


Figure 2.4: A directed, capacitated network.

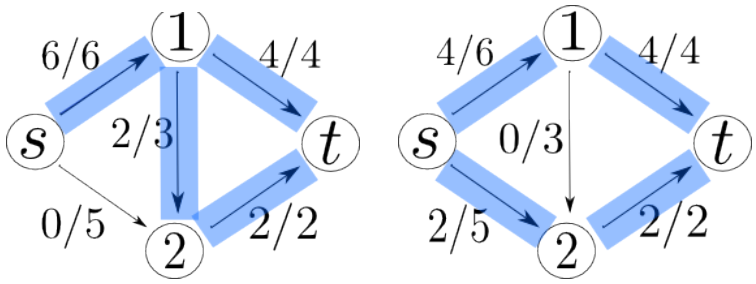


Figure 2.5: Two ways to send a flow through the network, giving the maximum value of the flow.

a flow through the network form $s$ to $t$ with maximum value. There are more ways to do this, two of them are given in Figure 2.5. The value of the flow is 6.
If we construct a cut separating the source vertex $t$ from the other vertices we get the minimum cut with value 6. This is shown in Figure 2.6. This means that the value of the flow of 6 is the maximum value in this network.

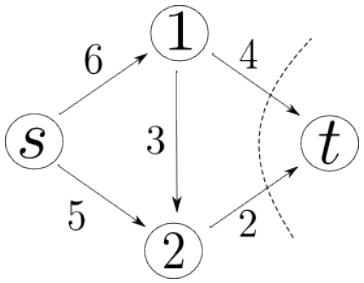


Figure 2.6: The minimum cut in the network.

**Ford-Fulkerson**

A method to find the maximum flow in a network is given by L.R. Ford and D.R. Fulkerson in 1956 [3]. For details on the maximum flow problem we refer the reader to the book *Flows in Networks* [4].
The method is based on augmenting paths. An augmenting path is the result of altering an existing path in such a way that in the new path the value of the flow is increased. This means that a flow in a network is maximal if there exists no flow augmenting path.
The algorithm to find the maximal flow consists of the following steps:

1. Start with a flow of value zero for all edges.
2. Find a path between the sink and the source vertex.
3. Search in this path for the edge with minimum residual capacity, i.e. the remaining capacity after sending the flow through the path.
4. Increase the value of the flow in this path with the value of this minimum residual capacity.

5. Repeat step 2-4 until there are no more paths possible.

At the end the value of the maximum flow is found.

#### Computing the Held-Karp bound

Using the techniques presented in this subsection the algorithm to solve (2.3) looks as follows:

1. Find a matrix $X$ that satisfies the constraints of the body (2.4).

2. For this matrix $X$, representing the adjacency of a graph $G$, search for the maximal flow in $G$ according to the Ford-Fulkerson method.

3. If the value of the maximal flow is greater than or equal to two, reject the solution matrix $X$ by cutting off the part of the polytope containing $X$.

4. Repeat these steps until the value of the maximal flow is less than two.

The solution matrix $X^*$ found in this way satisfies the constraints of the Held-Karp relaxation (2.3).

## 2.3 Van der Veen bound

Van der Veen proposed another way to calculate a lower bound to the symmetric circulant TSP [24]. To obtain the Van der Veen bound we consider a symmetric circulant $n \times n$ matrix $D$, i.e. a matrix of the form (2.2). Here the first $\lfloor \frac{n}{2} \rfloor + 1$ elements of the first row are distinct. Let $r = (r_0, \ldots, r_{\lfloor \frac{n}{2} \rfloor})$ be the vector consisting of these elements. In this way the vector $r$ covers all distinct entries of the matrix $D$. We use a permutation denoted $\phi$ to sort the values of $r$ in ascending order. Since we are dealing with the TSP and we assumed that $r_0 = 0$, this results in $\phi(0) = 0$. We give a short example for a better understanding of the definition of $\phi$.

#### Example

Suppose we have $n = 10$ and $r = (0, 1, 5, 8, 2)$. The permutation $\phi$ sorts the values of $r$ in ascending order, resulting in $r_\phi = (0, 1, 2, 5, 8)$. The permutation itself contains the following values: $\phi(r) = (0, 1, 4, 2, 3)$.

Van der Veen shows that we can use this permutation $\phi$ to construct a Hamiltonian tour, see [24]. To create this Hamiltonian tour we search for the smallest integer $l$ such that the greatest common divisor between $n$ and $\phi(1), \ldots, \phi(l)$ is one, i.e. $\gcd(n, \phi(1), \ldots, \phi(l)) = 1$. For this we introduce a new notation:

$$\texttt{GCD}(\phi(k)) := \gcd(\texttt{GCD}(\phi(k-1)), \phi(k)), \quad k = 1, \ldots, d, \tag{2.6}$$

where $\texttt{GCD}(\phi(0)) := n$.

With this new notation it follows that $\texttt{GCD}(\phi(l)) = 1$ as we wanted. This is indeed the case since:

$$\begin{aligned}\texttt{GCD}(\phi(l)) &= \gcd(\texttt{GCD}(\phi(l-1)), \phi(l)) \\ &= \gcd(\gcd(\texttt{GCD}(\phi(l-2)), \phi(l-1)), \phi(l)) \\ &= \gcd(n, \phi(1), \ldots, \phi(l)) \\ &= 1.\end{aligned}$$

The lower bound for the SCTSP is the length of the tour visiting each vertex ones and returning to the starting vertex. Theorem 7.4.2. in [24] gives the following formula to get a lower bound for the SCTSP:

$$VdV := \sum_{i=1}^{l} \{(\texttt{GCD}(\phi(i-1)) - \texttt{GCD}(\phi(i)))r_{\phi(i)}\} + r_{\phi(l)}, \tag{2.7}$$

where the last part $r_{\phi(l)}$ is the weight of the edge that closes the path obtained by $\sum_{i=1}^{l}\{(\texttt{GCD}(\phi(i-1)) - \texttt{GCD}(\phi(i)))r_{\phi(i)}\}$.
Observe that the Van der Veen bound can be computed in linear time. This has a big advantage with respect to the Held-Karp bound from (2.3).

## 2.4 Numerical comparisons

In this section we compare numerically the Held-Karp bound and the Van der Veen bound for the SCTSP. As proved by De Klerk and Dobre in [14], the Held-Karp bound is as least as tight as the Van der Veen bound for the symmetric circulant TSP. The aim of this section is to show that the Held-Karp bound does not dominate the Van der Veen bound if we take higher dimensional symmetric circulant matrices.
There are cases in which the heuristic proposed by Van der Veen gives the optimal value of the SCTSP in polynomial time. The following polynomial solvable cases are presented in detail in Chapter 7 of [24]:

- $n$ is prime,
- $\texttt{GCD}(\phi(1)) = 1$,
- $n = p^2$, where $p \geq 2$ is prime,
- $\texttt{GCD}(\phi(l-1)) = 2$,
- $\texttt{GCD}(\phi(l-1)) = d$ and both $\phi(l-1)$ and $\phi(l)$ are odd,
- $l = 2$, $\frac{n}{\texttt{GCD}(\phi(1))}$ is odd and $\texttt{GCD}(\phi(1)) \geq \frac{n}{\texttt{GCD}(\phi(1))} - 1$,
- $l = 2$, $\texttt{GCD}(\phi(l-1)) \leq 6$ and $\frac{r_{\phi(l+1)} - r_{\phi(l)}}{r_{\phi(l)} - r_{\phi(l-1)}} \geq \texttt{GCD}(\phi(l-1)) - 2$.

**Example**

Suppose we have the following $n \times n$ matrix:

$$M1 = \begin{pmatrix} 0 & 20 & 19 & 18 & 17 & 16 & 15 & 16 & 17 & 18 & 19 & 20 \\ 20 & 0 & 20 & 19 & 18 & 17 & 16 & 15 & 16 & 17 & 18 & 19 \\ 19 & 20 & 0 & 20 & 19 & 18 & 17 & 16 & 15 & 16 & 17 & 18 \\ 18 & 19 & 20 & 0 & 20 & 19 & 18 & 17 & 16 & 15 & 16 & 17 \\ 17 & 18 & 19 & 20 & 0 & 20 & 19 & 18 & 17 & 16 & 15 & 16 \\ 16 & 17 & 18 & 19 & 20 & 0 & 20 & 19 & 18 & 17 & 16 & 15 \\ 15 & 16 & 17 & 18 & 19 & 20 & 0 & 20 & 19 & 18 & 17 & 16 \\ 16 & 15 & 16 & 17 & 18 & 19 & 20 & 0 & 20 & 19 & 18 & 17 \\ 17 & 16 & 15 & 16 & 17 & 18 & 19 & 20 & 0 & 20 & 19 & 18 \\ 18 & 17 & 16 & 15 & 16 & 17 & 18 & 19 & 20 & 0 & 20 & 19 \\ 19 & 18 & 17 & 16 & 15 & 16 & 17 & 18 & 19 & 20 & 0 & 20 \\ 20 & 19 & 18 & 17 & 16 & 15 & 16 & 17 & 18 & 19 & 20 & 0 \end{pmatrix}. \tag{2.8}$$

The size of $M1$ is $12 \times 12$, so $n = 12$ and $d = \frac{12}{2} = 6$. The distinct numbers in $M1$ are represented by the vector $r = (0, 20, 19, 18, 17, 16, 15)$. The permutation $\phi$ has the following values: $\phi = (0, 6, 5, 4, 3, 2, 1)$. We can obtain the value of $l$ by computing the values of $\texttt{GCD}$. This gives:

- $\texttt{GCD}(\phi(1)) = gcd(n, \phi(1)) = gcd(12, 6) = 6$,
- $\texttt{GCD}(\phi(2)) = gcd(\texttt{GCD}(\phi(1)), \phi(2)) = gcd(6, 5) = 1$.

Since $\texttt{GCD}(\phi(2)) = 1$, we have that $l = 2$.
Now we have all the necessary information to see that $M1$ does not fall under any of the seven cases presented before:

- $n = 12$ is not prime,
- $\texttt{GCD}(\phi(1)) = 2 \neq 1$,
- $12 \neq p^2$, with $p$ prime,
- $\texttt{GCD}(\phi(l-1)) = 6 \neq 2$.
- $\texttt{GCD}(\phi(l-1)) = \texttt{GCD}(\phi(1)) = 6 = d$, but $\phi(l-1)$ is even and $\phi(l)$ is odd,
- $l = 2$, $\frac{n}{\texttt{GCD}(\phi(1))} = \frac{12}{6} = 2$ is even and $6 = \texttt{GCD}(\phi(1)) \geq \frac{n}{\texttt{GCD}(\phi(1))} - 1 = 1$,
- $l = 2$, $\texttt{GCD}(\phi(l-1)) = 6 \leq 6$ and $\frac{r_{\phi(l+1)} - r_{\phi(l)}}{r_{\phi(l)} - r_{\phi(l-1)}} = \frac{17-16}{16-15} = 1 \not\geq 4 = \texttt{GCD}(\phi(l-1)) - 2$.

The matrix $M1$ does not fall under any of the cases given above. This means that we can not say that the problem given by $M1$ is polynomially solvable.

We make use of instances that avoid the polynomial solvable instances. The numerical comparisons conducted by De Klerk and Dobre in [14] showed an equality between the Held-Karp bound and the Van der Veen bound for matrices up to a size of $81 \times 81$ and with a value of $l$ up to 4. Theoretically they proved that the Held-Karp bound is as least as tight as the Van der Veen bound. By taking instances with higher dimensions, i.e. up to 250, and with as much as possible distinct values of $l$, we try to find a strict inequality between the HK bound and the VdV bound.

| name | n | VdV | HK | number of cuts | distinct $l$ | computation time (s) Held-Karp |
|---|---|---|---|---|---|---|
| Dtot6000_250_L8 | 250 | 7875 | 7875 | 28 | 2 | 3,395 |
| Dtot1000_81_L4 | 81 | 534 | 534 | 20 | 2 | 65 |
| Dtot1000_81_L3 | 81 | 957 | 957 | 21 | 2 | 69 |
| Dtot1000_81_L5 | 81 | 396 | 396 | 20 | 2 | 63 |
| Dtot1000_87_L6 | 87 | 843 | 843 | 3 | 1 | 10 |
| Dtot1000_108_L8 | 108 | 519 | 519 | 13 | 2 | 71 |
| Dtot100_90_L7 | 90 | 360 | 360 | 180 | 3 | 2,052 |
| zelf216_8 | 216 | 1476 | 1476 | 355 | 5 | 104,400 |
| zelf144_8_5 | 144 | 288 | 288 | 566 | 5 | 135,443 |
| zelf200_5_3 | 200 | 268 | 268 | 328 | 3 | 59,596 |
| zelf168_5_4 | 168 | 322 | 322 | 291 | 4 | 27,119 |
| zelf135_6_3 | 135 | 325 | 325 | 488 | 3 | 72,066 |
| zelf128_7_6 | 128 | 254 | 254 | 220 | 6 | 6,927 |
| zelf180_7_4 | 180 | 462 | 462 | 90 | 4 | 3,775 |
| zelf90_4_3 | 90 | 250 | 250 | 202 | 3 | 2,076 |
| zelf180_6_5 | 180 | 424 | 424 | 338 | 5 | 33,204 |
| zelf128_6_4 | 128 | 422 | 422 | 19 | 4 | 126 |

Table 2.1: Numerical comparison of the lower bounds from Section 2.2 and 2.4 for symmetric circulant matrices.

### 2.4.1 Numerical results

The LP problems were solved using Matlab toolbox Yalmip together with the optimization solver Sedumi. The computations were carried on a standard Intel Core PC. The results are presented in Table 2.1.

Some remarks on Table 2.1:

- The Held-Karp bound and the Van der Veen bound are equal for all instances in the table.
- We have tried matrices with higher dimensions and a higher number of distinct $l$, but the computation of the Held-Karp bound ran into numerical problems, most likely due to memory issues.

The Matlab codes that are used for the computations are attached in Appendix D end Appendix E. The instances from Table 2.1 are available online at:

`http://www.math.rug.nl/~dobre/circulantMarTSP.tar.gz.`

**Conclusion**

Based on the numerical results we conjecture that the Held-Karp bound and the Van der Veen bound are equal. The Van der Veen algorithm makes use of the properties of the symmetric circulant matrices providing results for the SCTSP and can be computed in linear time. The Held-Karp relaxation is written for the general TSP and has a complex algorithm

that can be computed in polynomial time. Since both algorithms provide equal results and the computation time of the Van der Veen algorithm is much faster than the computation time of the Held-Karp algorithm, we suggest the use of the Van der Veen algorithm for the symmetric circulant TSP.

# Chapter 3

# Heuristics and tours

The aim of this chapter is to develop an algorithm that enables us to construct a tour out of the solution matrix of the LP relaxation given in Section 1.3 and the solution matrix of the SDP relaxation as is described in Section 1.4. Then we compare the length of the obtained tours with the tour lengths of three heuristics: the nearest-neighbor, the farthest insertion method and the method of biased random sampling. Next to this we compare the lower bound of the LP and SDP relaxation with the constructed tour value from our heuristic.

## 3.1 Heuristics

We first introduce the three heuristics. We describe how they construct a tour and then we provide a short example. The first two heuristics, namely the nearest-neighbor and the farthest insertion method, are deterministic in choosing an edge to be part of the tour. The method of biased random sampling is based on probability theory when doing the choice mentioned above.
We work with the symmetric Traveling Salesman Problem. Therefore the distance between two vertices is the same independent of the direction we take. This means that given the vertices $i$ and $j$ of the set of vertices $V$, the distance from vertex $i$ to vertex $j$ is equal to the distance from vertex $j$ to vertex $i$, hence the distance matrix $D$ is symmetric. If this is not the case we can transform the asymmetric TSP into a symmetric TSP with twice the number of vertices. A description of how to transform an asymmetric Traveling Salesman Problem into a symmetric one can be found in the paper of Jonker and Volgenant [10].

### 3.1.1 Nearest-neighbor

Recall from Section 1.6.1 that the nearest-neighbor heuristic consists of three steps [16]:

1. Start at an arbitrary vertex.
2. Consider the unvisited vertices in the neighborhood of the current vertex and pick the nearest neighbor. Repeat this step until all vertices are visited.
3. Return to the starting vertex to obtain a tour.

The complexity of the algorithm is $O(n^2)$. We give an example to illustrate the working of the nearest-neighbor algorithm.

**Example of nearest-neighbor**

Suppose 5 vertices are given with the following symmetric distance matrix:

$$D = \begin{pmatrix} 0 & 2 & 4 & 1 & 6 \\ 2 & 0 & 5 & 4 & 7 \\ 4 & 5 & 0 & 2 & 1 \\ 1 & 4 & 2 & 0 & 8 \\ 6 & 7 & 1 & 8 & 0 \end{pmatrix}. \tag{3.1}$$

If two vertices are not connected the distance is set infinity. Since this is not the case all vertices are connected to each other, i.e. we have a complete graph. Suppose we start with vertex 1. The neighborhood of the first vertex includes the other vertices $2, 3, 4$ and 5. The nearest vertex is vertex 4. This means that we connect vertex 1 to vertex 4. This is the start of our tour.
From vertex 4 we go to vertex 3 and from vertex 3 we go to vertex 5. The nearest unvisited vertex to vertex 5 is vertex 2. After including vertex 2 in our path we have visited all the vertices. This means that we return to our starting vertex 1 to close the path and obtain a tour. Our final tour is: $(1, 4, 3, 5, 2)$ and this notation means that the first and the last vertex are also connected. The total length of the tour is: $1 + 2 + 1 + 7 + 2 = 13$.

Suppose we do not start with vertex 1, but with vertex 3. This gives the following tour: $(3, 5, 1, 4, 2)$, with length: $1+6+1+4+5 = 17$. This illustrates that the starting vertex does influence the length of the tour. A way of getting a better tour is repeating the algorithm $n$ times, each time with an other vertex as starting vertex. The complexity of the algorithm is then $O(n^3)$.
The main drawback of the nearest-neighbor heuristic is that the edge that closes the path is not always a good one. If we work with constructing subtours this problem might not occur. Therefore in the next subsection we will look at a heuristic based on subtours.

### 3.1.2 Farthest insertion

Insertion methods start with a small subtour and increase this tour by adding the remaining vertices in a prescribed manner. There are several ways to add the remaining vertices. We describe here the farthest insertion method. For details see [2]. The steps for this heuristic are the following:

1. Start with an initial tour consisting of two vertices containing the highest edge value.
2. For each uninserted vertex $v$ compute the minimum distance $d$ between $v$ and the vertices in the tour.
3. Choose the vertex for which $d$ is maximal and insert this vertex in the tour in such a way that the length of the new obtained tour is minimal.
4. Repeat the last two steps until all vertices are inserted in the tour.

The complexity of the algorithm is $O(n^2)$.
Farthest insertion works best when the entries in the matrix $D$ represent Euclidean distances. The example given below is not Euclidean, however to see the difference in functioning relative to the nearest-neighbor we use the same distance matrix $D$ to describe the working of the farthest insertion method.

**Example of farthest insertion**

Suppose we have the same distance matrix $D$ as used in the previous example.

$$D = \begin{pmatrix} 0 & 2 & 4 & 1 & 6 \\ 2 & 0 & 5 & 4 & 7 \\ 4 & 5 & 0 & 2 & 1 \\ 1 & 4 & 2 & 0 & 8 \\ 6 & 7 & 1 & 8 & 0 \end{pmatrix}. \tag{3.2}$$

We start the tour with the two vertices defining the edge with highest distance. In the matrix $D$ the edge between the vertices 4 and 5 is 8, the highest value of the matrix. Therefore our starting vertices are 4 and 5. For the uninserted vertices $1, 2$ and 3 we compute the minimum distance $d$ to vertex 4 and 5. We get the following values $D(1,4) = 1$, $D(1,5) = 6$, $D(2,4) = 4$, $D(2,5) = 7$, $D(3,4) = 2$ and $D(3,5) = 1$. The minimum distances between the vertices $1, 2$ and 3 and the vertices of the tour are respectively: $d = 1$, $d = 4$ and $d = 1$. The maximum value of $d$ is 4. This is the edge between vertex 2 and 4. We insert vertex 2 in the tour. Since our tour consisted of only 2 vertices, the next vertex is automatically inserted between these two vertices. Our new tour is thus $(4, 5, 2)$.
For the uninserted vertices 1 and 3 we compute the minimal distances to the vertices of the tour. This results in $d = 1$ for vertex 1 and $d = 1$ for vertex 3. These values are equal and we may choose at random among them. Thus we choose vertex 1 to insert in the tour. Now we have to know at which place we have to insert the vertex in order to create a minimal new tour. There are three possibilities for the new tour: $(1, 4, 5, 2), (4, 1, 5, 2)$ or $(4, 5, 1, 2)$. For each of these possibilities we compute the total distance of the tour. For the tour $(1, 4, 5, 2)$ the total distance is: $D(1,4) + D(4,5) + D(5,2) + D(2,1) = 1 + 8 + 7 + 2 = 18$. In the same manner we compute the total distance for the second and third tour resulting in respectively 18 and 20. This means that there are two tours resulting in the lowest value of the tour length. We choose the tour: $(1, 4, 5, 2)$.
There is just one uninserted vertex left: vertex 3. The only thing we have to do is decide at what place in the tour we have to insert this vertex. Placing vertex 3 between respectively vertex 1 and 4, 4 and 5, 5 and 2 and 2 and 1 results in four different tours with the total distance values: $23, 13, 17$ and 25. In this way the optimal tour found by using farthest insertion is the tour $(1, 4, 3, 5, 2)$ with the value 13.

The farthest insertion method and the nearest-neighbor method result in the same tour for this example. For Euclidean distance matrices it turns out that in most cases the farthest insertion method gives better results than the nearest-neighbor method, see also [20].

### 3.1.3 Biased random sampling

Biased random sampling is based on the so called generic partitioning. In the case of the TSP the generic partitioning works in the following way: start at an arbitrary city. Then pick the next city randomly. This means that to every city not yet visited an equal probability of being selected is assigned. In this way every tour is possible. A disadvantage of generic partitioning is the fact that the objective function, i.e. minimizing the length of the tour, is not taken into account.
Biased random sampling is an extension of generic partitioning in order to take the objective function into account. In biased random sampling as we describe below the probability has

been biased in such a way that the probability of the city to be chosen next is higher when the distance between this city and the already placed city is smaller, see [12]. The steps of the biased random sampling method are the following:

1. Start at an arbitrary vertex.

2. For all vertices that are not yet in the tour compute the weights $w$ by using the distance between the last vertex in the tour and the unused vertices. Normalize these weights, to get $\tilde{w}$.

3. Take a randomly generated number $u$ between zero and one out of a uniform distribution.

4. Compare $u$ with the computed weights and select the next vertex by the selecting procedure (3.4).

5. Repeat step 2-4 until there is one vertex left. Insert this vertex in the tour.

The weight $w$ between the last vertex in the tour, city $j_t$, and the $m$ unused vertices $j_i$, for $i = 1, \ldots, m$ is given by:

$$w(j_t, j_i) = \frac{(D(j_t, j_i))^{-1}}{\sum_{h=1}^{m} (D(j_t, j_h))^{-1}}, \tag{3.3}$$

with $D$ being the distance matrix.
The selecting procedure for picking the next vertex in the tour is the following: if for vertex $j_{i*}$ it holds that

$$\sum_{p=1}^{i*} \tilde{w}(j_t, j_p) \leq u \leq \sum_{p=1}^{i*+1} \tilde{w}(j_t, j_p), \tag{3.4}$$

then vertex $j_{i*}$ becomes the next vertex in the tour. If there is no vertex satisfying this inequality, then choose the first of the unused vertices to become the next vertex in the tour. The complexity of the algorithm is $O(n^2)$.

**Example of biased random sampling**

To see the difference in outcome of the three heuristics, we make use of the same $5 \times 5$ matrix $D$ to describe the working of biased random sampling. So:

$$D = \begin{pmatrix} 0 & 2 & 4 & 1 & 6 \\ 2 & 0 & 5 & 4 & 7 \\ 4 & 5 & 0 & 2 & 1 \\ 1 & 4 & 2 & 0 & 8 \\ 6 & 7 & 1 & 8 & 0 \end{pmatrix}. \tag{3.5}$$

We take the first vertex as the starting vertex. For the vertices 2-5 we compute the weights $w$. This gives:

$$w(1,2) = \frac{D(1,2)^{-1}}{(D(1,2)+D(1,3)+D(1,4)+D(1,5))^{-1}} = \frac{2^{-1}}{(2+3+1+6)^{-1}} = \frac{0.5}{1/12} = 6,$$
$$w(1,3) = \frac{D(1,3)^{-1}}{(D(1,2)+D(1,3)+D(1,4)+D(1,5))^{-1}} = \frac{4^{-1}}{(2+3+1+6)^{-1}} = \frac{0.25}{1/12} = 3,$$
$$w(1,4) = \frac{D(1,4)^{-1}}{(D(1,2)+D(1,3)+D(1,4)+D(1,5))^{-1}} = \frac{1^{-1}}{(2+3+1+6)^{-1}} = \frac{1}{1/12} = 12,$$
$$w(1,5) = \frac{D(1,5)^{-1}}{(D(1,2)+D(1,3)+D(1,4)+D(1,5))^{-1}} = \frac{6^{-1}}{(2+3+1+6)^{-1}} = \frac{1/6}{1/12} = 2.$$

Normalizing the weights gives: $\tilde{w}(1,2) = 6/23$, $\tilde{w}(1,3) = 3/23$, $\tilde{w}(1,4) = 12/23$, $\tilde{w}(1,5) = 2/23$. The next step is taking a randomly generated number $u$ between 0 and 1. Assume we get $u = 0.54$. We compare the weights with $u$ by making use of the selection procedure (3.4). This means that we make a summation of the weights and take the vertex that satisfies the inequality $\sum_{p=1}^{i*} \tilde{w}(1,j_p) \leq u \leq \sum_{p=1}^{i*+1} \tilde{w}(1,j_p)$. In our case it turns out that: $\tilde{w}(1,2) + \tilde{w}(1,3) \leq u \leq \tilde{w}(1,2) + \tilde{w}(1,3) + \tilde{w}(1,4)$ since this is $0.39 \leq 0.54 \leq 0.91$. This means that vertex 3 becomes the next vertex in the tour.
The unused vertices are now 2,4 and 5. Our current vertex is vertex 3. We compute the weights resulting in: $w(3,2) = 1.6$, $w(3,4) = 4$, $w(3,5) = 8$. So the normalized weights are: $\tilde{w}(3,2) = 2/17$, $\tilde{w}(3,4) = 5/17$, $\tilde{w}(3,5) = 10/17$. We take an uniformly generated number $u = 0.31$. It turns out that $\tilde{w}(3,2) \leq u \leq \tilde{w}(3,2) + \tilde{w}(3,4)$, since $0.12 \leq 0.51 \leq 0.41$. This means that we take vertex 2 as the next vertex in the tour.
Now we have two vertices left: vertex 4 and vertex 5. Our current vertex is vertex 2. The weights are: $w(2,4) = 11/4$, $w(2,5) = 11/7$. Therefore the normalized weights are: $\tilde{w}(2,4) = 7/11$, $\tilde{w}(2,5) = 4/11$. We take a new uniformly generated number $u = 0.63$. Since $\tilde{w}(2,4) = 0.64$, it means there is no sum of vertices less than or equal to $u$. Thus we take the first unused vertex, vertex 4 to insert in our tour. This means that we have only one vertex left. This vertex closes the tour. So our total tour becomes: $(1,3,2,4,5)$ and the tour length is: $4+5+4+8+6 = 27$.

Each execution of the algorithm different randomly generated numbers are used. This means that the outcome of the random biased sampling will not be the same each time that we run the program with the same starting vertex. In this example the total tour length of 27 is far higher than the outcome of the nearest-neighbor and the farthest insertion method. If we had got different random generated numbers it would have been possible that we attained a tour as short as 13.
As with the nearest-neighbor algorithm the starting vertex also influences the final tour. A way to get a better possible tour using biased random sampling with the given vector of generated numbers is to repeat the algorithm $n$ times, each time with a different starting vertex. The complexity of the algorithm is then $O(n^3)$.

### 3.1.4 Euclidean symmetric TSP

In the rest of this chapter we will work with the Euclidean symmetric Traveling Salesman Problem unless otherwise stated, since the farthest insertion method works best for this

problem. Additionally, it enables us to draw pictures of the tours.

### $6 \times 6$ matrix

We start with the following points in $\mathbb{R}^2$ which define the vertices of the graph: $(1,2)$, $(3,1)$, $(3,2)$, $(5,2)$, $(5,4)$ and (2,5). One obtains the corresponding distance matrix:

$$D = \begin{pmatrix} 0 & 2.2361 & 2 & 4 & 4.4721 & 3.1623 \\ 2.2361 & 0 & 1 & 2.2361 & 3.6056 & 4.1231 \\ 2 & 1 & 0 & 2 & 2.8284 & 3.1623 \\ 4 & 2.2361 & 2 & 0 & 2 & 4.2426 \\ 4.4721 & 3.6056 & 2.8284 & 2 & 0 & 3.1623 \\ 3.1623 & 4.1231 & 3.1623 & 4.2426 & 3.1623 & 0 \end{pmatrix}. \tag{3.6}$$

To obtain the adjacency matrices of the tours given by the heuristics we first construct the tours. We compute the tours for the three heuristics described above: the nearest-neighbor insertion, the farthest insertion and biased random sampling.

*Nearest-neighbor insertion* We start with the first vertex. If we take each time the nearest vertex, this results in the tour: $(1,3,2,4,5,6)$, with tour length: 13.5606. This is shown in Figure 3.1.

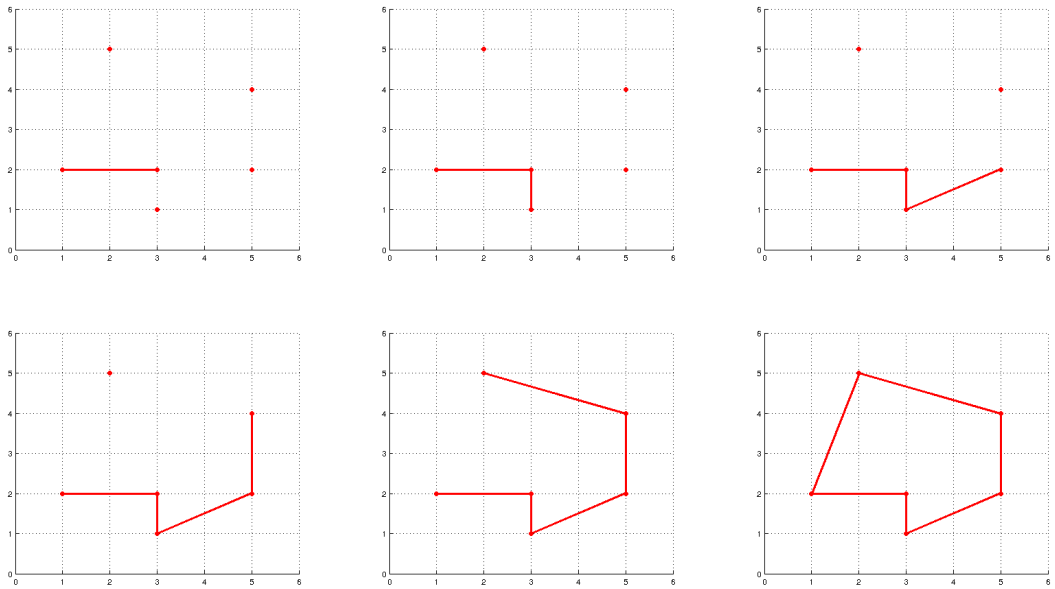

Figure 3.1: Growing tour with nearest insertion.

*Farthest insertion* We start with the two vertices containing the edge with the highest distance. These are the vertices 1 and 5. For the uninserted vertices we compute the minimum distance to the starting vertices. This gives $\min(D(2,1), D(2,5)) = 2.2361$, $\min(D(3,1), D(3,5)) = 2$, $\min(D(4,1), D(4,5)) = 2$ and $\min(D(6,1), D(6,5)) = 3.1623$. The maximum value is 3.1623,

the distance between both vertex 6 and 1 and vertex 6 and 5. As we are looking for the third vertex in the tour, we do not have to choose and insert the vertex next to vertex 1 and 5. This gives the new tour: $(1, 5, 6)$.

Again we compute the minimum distance between the uninserted vertices and the vertices in the tour. This results in $\min(D(2,1), D(2,5), D(2,6)) = 2.2361$, $\min(D(3,1), D(3,5), D(3,6)) = 2$ and $\min(D(4,1), D(4,5), D(4,6)) = 2$. The maximum value is 2.2361, corresponding to $D(2,1)$. We insert vertex 2 in the tour. There are three possibilities for the new tour: $(1, 2, 5, 6)$, $(1, 5, 2, 6)$ and $(1, 5, 6, 2)$. For each tour we compute the total distance, resulting in respectively 12.1662, 15.3631 and 13.9936. The smallest tour is reached when we take the first option. So the new tour is: $(1, 2, 5, 6)$.

Computing the minimum distances between the uninserted vertices and the vertices in the tour, taking the maximum value and compute the position in the tour for the last two uninserted vertices results in the tour: $(1, 2, 4, 5, 6)$, with tour length: 12.7967.

For the last uninserted vertex 3 we only have to compute the position in the tour. This results in the tour: $(1, 3, 2, 4, 5, 6)$, with tour length: 13.5606. This is shown in Figure 3.2.

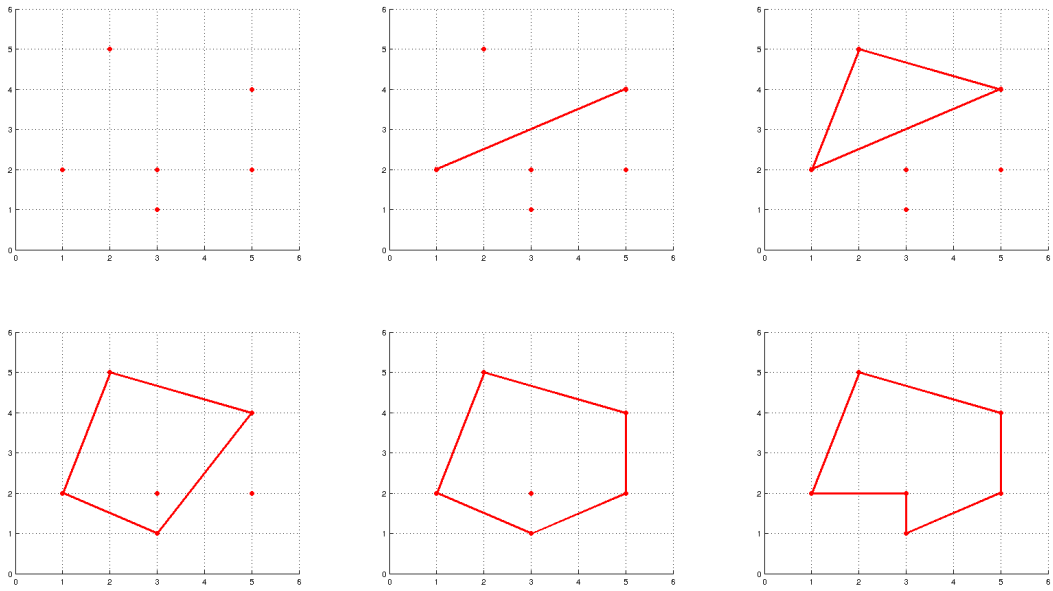

Figure 3.2: Growing tour with farthest insertion.

*Biased random sampling* We take again the first vertex as the starting vertex. For the vertices

2-6 we compute the weights, this gives:

$$w(1,2) = \frac{D(1,2)^{-1}}{(D(1,2)+D(1,3)+D(1,4)+D(1,5)+D(1,6))^{-1}} = \frac{0.4472}{1/15.8705} = 7.097,$$
$$w(1,3) = \frac{D(1,3)^{-1}}{(D(1,2)+D(1,3)+D(1,4)+D(1,5)+D(1,6))^{-1}} = \frac{0.5}{1/15.8705} = 7.935,$$
$$w(1,4) = \frac{D(1,4)^{-1}}{(D(1,2)+D(1,3)+D(1,4)+D(1,5)+D(1,6))^{-1}} = \frac{0.25}{1/51.8705} = 3.968,$$
$$w(1,5) = \frac{D(1,5)^{-1}}{(D(1,2)+D(1,3)+D(1,4)+D(1,5)+D(1,6))^{-1}} = \frac{0.2236}{1/15.8705} = 3.549,$$
$$w(1,6) = \frac{D(1,6)^{-1}}{(D(1,2)+D(1,3)+D(1,4)+D(1,5)+D(1,6))^{-1}} = \frac{0.3162}{1/15.8705} = 5.019.$$

We normalize these weights to get $\tilde{w}(1,2) = 0.257$, $\tilde{w}(1,3) = 0.288$, $\tilde{w}(1,4) = 0.144$, $\tilde{w}(1,5) = 0.129$ and $\tilde{w}(1,6) = 0.182$. We take $u = 0.8909$. Since $\tilde{w}(1,2) + \tilde{w}(1,3) + \tilde{w}(1,4) + \tilde{w}(1,5) = 0.818 \leq 0.8909$, we take vertex 5 as the next vertex. Again we start with computing the weights, but now for the vertices 2, 3, 4 and 6. Normalizing the weights gives: $\tilde{w}(5,2) = 0.1917$, $\tilde{w}(5,3) = 0.2443$, $\tilde{w}(5,4) = 0.3445$ and $\tilde{w}(5,6) = 0.2185$. Taking $u = 0.3342$ this gives that the next vertex in the tour is vertex 2.
For the remaining vertices 3, 4 and 6, we compute the weights, normalize them and take $u = 0.6987$. Since $\tilde{w}(2,3) = 0.5918$ and $\tilde{w}(2,4) = 0.2647$, the sum of these two normalized weights is greater than $u$, so we take vertex 3 as the next vertex in the tour.
Now we only have two vertices left. Their normalized weights are $\tilde{w}(3,4) = 0.6126$ and $\tilde{w}(3,6) = 0.3847$. We get $u = 0.1978$. Since none of the two weights is less than $u$, we take the first vertex as the next vertex in the tour. This means that our last remaining vertex closes the tour, resulting in the total tour of $(1,5,2,3,4,6)$ with tour length 18.4826. This is shown in Figure 3.3.
The three heuristics are written down in Matlab code in order to compare the heuristics numerically. For the nearest-neighbor and the biased random sampling the algorithm executes the code $n$ times in order to get a better result. The Matlab codes are attached in Appendix A.

## 3.2 Using linear and semidefinite programming for TSP

Recall that the linear programming relaxation resulting in the Held-Karp bound as given in Section 1.3 was defined as follows:

$$\begin{aligned} HK := \min\ & \frac{1}{2}\text{trace}(DX) \\ \text{s.t. } & Xe = 2e, \\ & \text{diag}(X) = 0, \\ & 0 \leq X \leq J, \\ & \sum_{i \in I, j \notin I} X_{ij} \geq 2 \quad \forall\ \emptyset \neq I \subset \{1,\ldots,n\}, \end{aligned} \tag{3.7}$$

where $D$ denotes the distance matrix, $e$ the all-ones vector and $J$ the all-ones matrix.

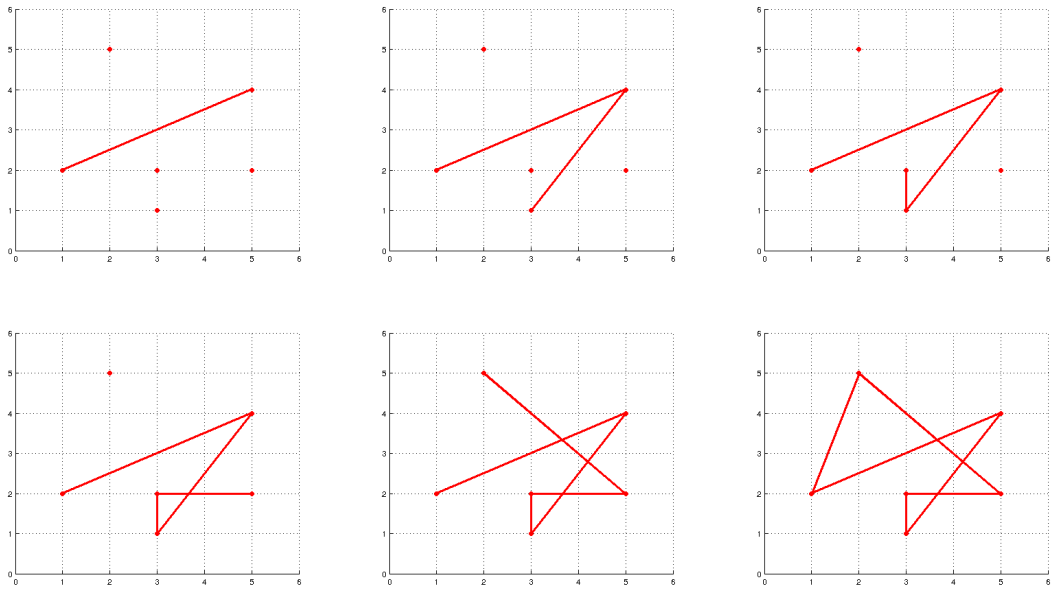

Figure 3.3: Growing tour with biased random sampling.

In Section 1.4 we introduced the following semidefinite programming relaxation for the Traveling Salesman Problem:

$$\begin{aligned} KPS := \min\ & \frac{1}{2}\text{trace}(DX^{(1)}) \\ \text{s.t. } & X^{(k)} \geq 0, \quad k = 1, \ldots, d \\ & \sum_{k=1}^{d} X^{(k)} = J - I, \\ & I + \sum_{k=1}^{d} \cos(\frac{2ki\pi}{n}) X^{(k)} \succeq 0, \quad i = 1, \ldots, d \\ & X^{(k)} \in \mathbb{S}^{n \times n}, \quad k = 1, \ldots, d, \end{aligned} \tag{3.8}$$

where $D$ denotes the distance matrix, $J$ the all-ones matrix and $d = \lfloor \frac{n}{2} \rfloor$.
De Klerk, Pasechnik, and Sotirov showed in [15] that this SDP provides a lower bound on the length of an optimal tour. Further they showed that for general TSP the Held-Karp bound (3.7) does not dominate the SDP bound (3.8) or vice versa. This means that for general TSP the lower bound obtained by solving the SDP program will be sometimes higher and other times lower than the lower bound obtained by the linear programming relaxation for the Traveling Salesman Problem from the Held-Karp bound.

### 3.2.1 Solution matrix $X$

The value of the tour, approximated using linear programming and semidefinite programming is a lower bound to the value of the optimal tour $T$. Since LP and SDP approaches weakened

the constraints to obtain an approximation to the problem, the entries of the solution matrix $X$ of (3.7) or (3.8) will not consist of only zeros and ones. Instead, the entries of the matrix $X$ will lie between zero and one. This means that we cannot directly construct a tour out of the solution matrix. This is in contrast to the heuristics presented before, because the outcome of the heuristics is a tour. In the following we aim to come up with a heuristic to round the values of $X$ in order to obtain a tour. To this end we begin by comparing the adjacency matrices of the nearest-neighbor method, the farthest insertion method and the biased random sampling with the solution matrices of the linear and semidefinite programming relaxations (3.7) and (3.8).

*Held-Karp and semidefinite programming* If we take the distance matrix (3.6) we can compute the solution matrices corresponding to the Held-Karp relaxation (3.7) and the semidefinite programming relaxation (3.8). The solution matrices denoted by $X$ are computed using Matlab toolbox Yalmip together with the optimization solver Sedumi. The results are:

$$X_{HK} = \begin{pmatrix} 0 & 0.4996 & 0.5004 & 0 & 0 & 1 \\ 0.4996 & 0 & 1 & 0.5004 & 0 & 0 \\ 0.5004 & 1 & 0 & 0.4996 & 0 & 0 \\ 0 & 0.5004 & 0.4996 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 \end{pmatrix} \tag{3.9}$$

and

$$X_{KPS} = \begin{pmatrix} 0 & 0.5000 & 0.5000 & 0 & 0 & 1 \\ 0.5000 & 0 & 1 & 0.4999 & 0 & 0 \\ 0.5000 & 1 & 0 & 0.5001 & 0 & 0 \\ 0 & 0.4999 & 0.5001 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}, \tag{3.10}$$

both with an objective value of 13.5606.
The three heuristics gave us three tours. Out of these tours we can construct the adjacency matrices. Since the tour using the first two heuristics was for this example the same, their adjacency matrices are equal:

$$X_{nearest} = X_{farthest} = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}. \tag{3.11}$$

The adjacency matrix of the tour using the biased random sampling heuristic is different:

$$X_{biased} = \begin{pmatrix} 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}. \tag{3.12}$$

Recall that the tour constructed with the use of the biased random sampling method is not always the same, depending on the uniform distributed number $u$. Repeating the biased random sampling method with the same distance matrix and the same starting vertex gives different tours, like $(1, 2, 4, 5, 3, 6)$, with tour length 15.6251 and $(1, 2, 3, 4, 5, 6)$ with tour length 13.5606.

### 3.2.2 Constructing a tour

The aim of the tour contructing is to obtain the adjacency matrix of a tour, i.e. a zero-one matrix, out of the solution matrix $X$ of (3.7) or (3.8). In some cases the solution matrix $X$ consists already of only zeros and ones, which means that we have an adjacency matrix of the tour corresponding to the problem. In other cases the entries of the matrix $X$ lie between zero and one. In that case we apply the construction method to the solution matrix $X$ to obtain the adjacency matrix of the tour.
The steps of the construction method are the following:

1. Start with row $k$ of the matrix. There are three possibilities:

   - The row contains two entries equal to one. Take the first of these entries as the next vertex. Mark the other entry as the last entry.
   - There is exactly one entry equal to one in the row. Take this entry $i$ as the next vertex. Search for the maximum value of the remaining entries in the row. Set this entry $j$ to one and set the remaining entries in the row to zero. Change the entries of the $k$th column to zero except the entries $i$ and $j$. Set $i$ and $j$ to one. Mark entry $j$ as the last entry.
   - The row contains no entries equal to one. Search for the two maximum values in the row. If there are more, take the first two. Set these entries $i$ and $j$ to one and set the remaining entries in the row to zero. Change the entries in the $k$th column to zero except the entries $i$ and $j$. Set these entries to one. Take the entry with the highest value, entry $i$, as the next vertex. Mark entry $j$ as the last entry.

2. In row $i$ search for the maximum value among the entries of row $i$, except for entry $k$. Take this entry $m$ as the next vertex. Set the other entries in the row zero. Also set the entries in column $i$ zero except the entries $k$ and $m$, change them to one. If the maximum value is equal to zero, take the first of the unused vertices as the next vertex $m$. Set $k = m$ and $m = i$. Repeat this step $n - 3$ times.

3. For the closing vertex set all entries in the row and column $i$ zero, except entry $k$ and the last entry $j$, change them to one.

This gives the constructed adjacency matrix $A$. Out of this matrix the tour and the corresponding tour length can be easily computed. The Matlab code is attached in Appendix E.

**Example**

Given the distance matrix (3.6), we show the construction method for the solution matrix of (3.7). Recall from Section 3.2.1 that we have the solution matrix:

$$X_{HK} = \begin{pmatrix} 0 & 0.4996 & 0.5004 & 0 & 0 & 1 \\ 0.4996 & 0 & 1 & 0.5004 & 0 & 0 \\ 0.5004 & 1 & 0 & 0.4996 & 0 & 0 \\ 0 & 0.5004 & 0.4996 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}. \tag{3.13}$$

We start with the first row. The sixth entry is equal to one, so this is our next vertex. We search for the maximum value of the remaining entries, getting the value 0.5004, corresponding to the third entry in the row. We mark this third entry as our last entry. We set the entries of the first row and the first column equal to zero except for the third and sixth entry. This gives:

$$\begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0.5004 & 0 & 0 \\ 1 & 1 & 0 & 0.4996 & 0 & 0 \\ 0 & 0.5004 & 0.4996 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}.$$

The next row is the sixth. Next to the first entry, the fifth entry is already equal to one. This means that we do not have to change the matrix and we go on to the fifth row. We find an entry equal to one on the fourth place. So the next row is the fourth. We search for the maximum value next to the fifth entry. This is 0.5004, corresponding to the second entry. We set the entries of the fourth row and column equal to zero except the second and the fifth entry which we set one. This gives:

$$\begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}.$$

We see that we have already an adjacency matrix. For the sake of clarity we complete our algorithm.
The last row we have to deal with is the second row. We only have to insert the closing edge. So we set all the entries in the second row and column equal to zero except the fourth entry and the last entry, entry three. This gives:

$$A_{HK} = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}. \tag{3.14}$$

Out of the adjacency matrix $A_{HK}$ we can construct the tour: $(1, 6, 5, 4, 2, 3)$ with tour length 13.5606.

### 3.2.3 Comparing the constructed tour value with the heuristics

Recall from Section 1.6.3 that the constructed tour value will be higher than or equal to the optimal value of the Traveling Salesman Problem. Similarly the values obtained by the farthest insertion, nearest insertion and biased random sampling will be higher than or equal to the optimal value. The lower bound for the TSP using SDP or LP will be lower than or equal to the optimal value of the TSP, since LP and SDP weakened the constraints to obtain a solution matrix $X$ to the problem.

The LP and SDP relaxations (3.7) and (3.8) were solved using Matlab toolbox Yalmip together with the optimization solver Sedumi. The value of the tour after applying the constructed tour method to the solution matrix $X$ of (3.7) and (3.8) are given by respectively HK tour and KPS tour. The optimum values of the instances were computed using the Concorde software for the Traveling Salesman Problem, available at

`http://neos.mcs.anl.gov/neos/solvers/co:concorde/TSP.html.`

The computations were carried on a standard Intel Core i5 PC. All values in the tables could be computed in a maximum of a few minutes. For Table 3.1 and Table 3.2 instances of the metric TSP are used. Instances of the general TSP are used for Table 3.3 and Table 3.4.

| name | n | HK tour | KPS tour | farthest | nearest | biased | optimum |
|---|---|---|---|---|---|---|---|
| coor1 | 11 | 311.7315 | 319.8047 | 311.7315 | 311.7615 | 460.3281 | 311 |
| coor2 | 13 | 318.4912 | 318.4912 | 318.4912 | 352.5902 | 548.8009 | 318 |
| coor3 | 18 | 371.7828 | 372.3932 | 371.7828 | 379.1632 | 695.9953 | 370 |
| coor4 | 19 | 360.9159 | 421.6753 | 373.8912 | 366.8610 | 829.7177 | 361 |
| coor5 | 20 | 318.6121 | 318.6121 | 318.6121 | 333.5798 | 819.8129 | 318 |
| coor6 | 25 | 438.4459 | 438.4459 | 463.5943 | 446.5927 | 1112.9 | 438 |
| coor7 | 14 | 367.7759 | 415.6957 | 369.9029 | 409.6497 | 715.4192 | 366 |
| coor8 | 14 | 220.3548 | 220.3548 | 220.3548 | 226.3588 | 241.4916 | 219 |
| coor9 | 6 | 13.5606 | 13.5606 | 13.5606 | 13.5606 | 14.7967 | 13 |
| coor10 | 8 | 1604.6 | 1604.6 | 1604.6 | 1605.3 | 1616.7 | 1604 |
| coor11 | 23 | 463.5329 | 463.5329 | 443.3704 | 472.9929 | 930.7689 | 442 |
| coor12 | 7 | 249.8463 | 249.8463 | 249.8463 | 260.6418 | 253.4189 | 250 |
| coor13 | 9 | 328.4321 | 328.4321 | 328.4321 | 344.9748 | 419.6906 | 329 |
| coor14 | 10 | 322.7712 | 322.7712 | 322.7712 | 322.7712 | 490.6260 | 323 |
| coor15 | 12 | 309.3584 | 309.3584 | 309.3584 | 309.7328 | 485.7474 | 309 |
| coor16 | 15 | 316.8109 | 316.8109 | 324.6346 | 340.9313 | 634.6920 | 316 |
| coor17 | 16 | 356.4716 | 356.4716 | 358.7312 | 378.2720 | 670.7334 | 355 |
| coor18 | 22 | 497.0969 | 532.2524 | 511.1667 | 540.3911 | 1060.7 | 497 |
| coor19 | 26 | 393.3237 | 393.3237 | 393.3237 | 416.2467 | 1131.4 | 392 |
| coor20 | 24 | 472.0170 | 468.9648 | 444.0807 | 459.24 | 1109.5 | 430 |

Table 3.1: Numerical comparison of the constructed tour value with the tour length of the heuristics for the metric TSP.

A few remarks on Table 3.1:

- The Concorde software rounded the coordinates of the instances before computing the optimum value. This explains for the instances *coor*12, *coor*13 and *coor*14 that the value of the optimal tour is higher than the constructed tour values and some values obtained by the three heuristics.
- Biased random sampling performs by far the worst for all instances, except for instance *coor*11, there nearest-neighbor performs worse.
- The value of the tour obtained by the construction method for the LP and SDP program appears to be close to the optimum value and performs often better than the farthest insertion and nearest-neighbor method.

| name | n | HK bound | HK tour | KPS bound | KPS tour |
|---|---|---|---|---|---|
| coor1 | 11 | 311.7315 | 311.7315 | 301.792 | 319.8047 |
| coor2 | 13 | 318.4912 | 318.4912 | 318.441 | 318.4912 |
| coor3 | 18 | 371.7828 | 371.7828 | 369.36 | 372.3932 |
| coor4 | 19 | 360.9159 | 360.9159 | 342.57 | 421.6753 |
| coor5 | 20 | 318.6121 | 318.6121 | 318.37 | 318.6121 |
| coor6 | 25 | 438.4459 | 438.4459 | 434.30 | 438.4459 |
| coor7 | 14 | 367.7759 | 367.7759 | 357.954 | 415.6957 |
| coor8 | 14 | 220.3548 | 220.3548 | 219.492 | 220.3548 |
| coor9 | 6 | 13.5606 | 13.5606 | 13.5606 | 13.5606 |
| coor10 | 8 | 1604.6 | 1604.6 | 1604.4 | 1604.6 |
| coor11 | 23 | 435.383 | 463.5329 | 431.91 | 463.5329 |
| coor12 | 7 | 249.8463 | 249.8463 | 247.9185 | 249.8463 |
| coor13 | 9 | 328.4321 | 328.4321 | 328.4321 | 328.4321 |
| coor14 | 10 | 322.7712 | 322.7712 | 322.485 | 322.7712 |
| coor15 | 12 | 309.3584 | 309.3584 | 309.3227 | 309.3584 |
| coor16 | 15 | 316.8109 | 316.8109 | 316.6287 | 316.8109 |
| coor17 | 16 | 356.4716 | 356.4716 | 355.733 | 356.4716 |
| coor18 | 22 | 497.0969 | 497.0969 | 488.4 | 562.2524 |
| coor19 | 26 | 393.3237 | 393.3237 | 393.3237 | 393.3237 |
| coor20 | 24 | 429.4686 | 472.0170 | 428.46 | 468.9648 |

Table 3.2: Numerical comparison of the lower bound using the relaxations (3.7) and (3.8) and the constructed tour value for the metric TSP.

In Table 3.2 it can be seen that for all instances except for the instances *coor*11 and *coor*20 the Held-Karp bound and the value of the associated constructed tour are equal. This means that in those cases the Held-Karp bound equals the optimum.
More computations showed that for small instances of the metric TSP the solution matrix $X$ to the Held-Karp relaxation (3.7) most of the time gave directly the adjacency matrix of a tour. After applying the constructed tour method the adjacency matrix remained the same. In these cases the Held-Karp bound equals the optimum.

A few remarks on Table 3.3:

| name | n | HK tour | KPS tour | farthest | nearest | biased | optimum |
|---|---|---|---|---|---|---|---|
| Krand1 | 9 | 157 | 157 | 244 | 165 | 257 | 157 |
| Krand2 | 12 | 236 | 236 | 256 | 190 | 439 | 180 |
| Krand3 | 9 | 147 | 147 | 167 | 164 | 336 | 147 |
| Krand4 | 13 | 315 | 315 | 312 | 291 | 641 | 267 |
| Krand5 | 15 | 270 | 367 | 319 | 295 | 603 | 270 |
| Krand6 | 16 | 223 | 223 | 361 | 299 | 751 | 223 |
| Krand7 | 16 | 169 | 169 | 227 | 169 | 547 | 169 |
| Krand8 | 16 | 321 | 316 | 317 | 348 | 634 | 230 |
| Krand9 | 18 | 312 | 312 | 409 | 387 | 819 | 312 |
| Krand10 | 20 | 132 | 132 | 233 | 212 | 645 | 132 |
| Krand11 | 22 | 335 | 335 | 438 | 283 | 782 | 278 |
| Krand12 | 23 | 359 | 211 | 350 | 271 | 843 | 211 |
| Krand13 | 21 | 443 | 321 | 415 | 328 | 862 | 244 |
| Krand14 | 8 | 281 | 192 | 240 | 178 | 270 | 178 |
| Krand15 | 17 | 281 | 281 | 294 | 217 | 723 | 198 |
| Krand16 | 19 | 171 | 171 | 198 | 186 | 717 | 171 |
| Krand17 | 19 | 255 | 255 | 386 | 304 | 712 | 255 |
| Krand18 | 21 | 195 | 195 | 325 | 298 | 917 | 195 |
| Krand19 | 22 | 289 | 173 | 334 | 274 | 1001 | 170 |
| Krand20 | 24 | 376 | 248 | 407 | 269 | 1007 | 245 |

Table 3.3: Numerical comparison of the constructed tour value with the tour length of the heuristics for the general TSP.

- The constructed tour value given by KPS tour performs best compared to the optimum for fourteen of the instances. Sometimes this value equals the value of HK tour and for instance $Krand7$ this is also equal to the value obtained by the nearest-neighbor algorithm. For five instances the nearest-neighbor algorithm performs best and for one instance the HK tour was the only one equal to the optimum.

- Biased random sampling, in the way we have implemented it, performs the worst compared to the optimum.

In Table 3.4 it can be seen that for half of the instances the Held-Karp bound and the corresponding constructed tour value are equal, which means that the Held-Karp bound gives the optimum. Next to this it can be seen that the Held-Karp bound performs better than the bound obtained by SDP, except for instance $Krand19$. This is consistent with the work of De Klerk, Pasechnik and Sotirov in [15] showing that for general TSP the Held-Karp bound does not dominate the SDP bound or vice versa.

The instances are available online at:

`http://www.math.rug.nl/~dobre/metricMarTSP.tar.gz`

and

`http://www.math.rug.nl/~dobre/randomMarTSP.tar.gz.`

| name | n | HK bound | HK tour | KPS bound | KPS tour |
|---|---|---|---|---|---|
| Krand1 | 9 | 157 | 157 | 157 | 157 |
| Krand2 | 12 | 180 | 236 | 178 | 236 |
| Krand3 | 9 | 147 | 147 | 147 | 147 |
| Krand4 | 13 | 263.5 | 315 | 261.7 | 315 |
| Krand5 | 15 | 270 | 270 | 270 | 367 |
| Krand6 | 16 | 223 | 223 | 223 | 223 |
| Krand7 | 16 | 169 | 169 | 169 | 169 |
| Krand8 | 16 | 229.5 | 321 | 229.1 | 316 |
| Krand9 | 18 | 312 | 312 | 312 | 312 |
| Krand10 | 20 | 132 | 132 | 131.1 | 132 |
| Krand11 | 22 | 277.667 | 335 | 277.6 | 335 |
| Krand12 | 23 | 210 | 359 | 210 | 211 |
| Krand13 | 21 | 244 | 443 | 243.9 | 321 |
| Krand14 | 8 | 177.5 | 281 | 177.5 | 192 |
| Krand15 | 17 | 198 | 281 | 197.2 | 281 |
| Krand16 | 19 | 171 | 171 | 171 | 171 |
| Krand17 | 19 | 255 | 255 | 255 | 255 |
| Krand18 | 21 | 195 | 195 | 194.1 | 195 |
| Krand19 | 22 | 169.5 | 289 | 169.6 | 173 |
| Krand20 | 24 | 242 | 376 | 241.2 | 248 |

Table 3.4: Numerical comparison of the lower bound using the relaxations (3.7) and (3.8) and the constructed tour value for the general TSP.

The Matlab files contain the distance matrix $K$, the coordinate matrix *coor*, the solution matrix $g$ of (3.7) and the solution matrix *sol* of (3.8).

### 3.2.4 Conclusion

The construction method, both for the LP and SDP relaxations (3.7) and (3.8), provides a useful addition to the existing heuristics. For the metric TSP our constructed tour heuristic provides the best results next to farthest insertion. For the general TSP our constructed tour heuristic provides again the best results, this time next to the nearest-neighbor heuristic. The value of the KPS tour is most of the time close to the optimum.
We knew from De Klerk, Pasechnik and Sotirov in [15] that the Held-Karp bound does not dominate the SDP bound or vice versa. Out of the numerical results we can conclude that also the Held-Karp tour does not dominate the KPS tour.

An open question remains to establish theoretical results on the ratio between the optimal value of the TSP and the value given by our heuristic.

# Appendix A

# Heuristics

## A.1 Adjacency

```
%Mariëlle Kruithof
%Adjacency matrix of a tour
%Input: tour of a graph.
%Output: adjacency matrix A of a tour.

function A=adjacency(tour)
n=length(tour);
A=zeros(n,n);
for i=1:n-1
    A(tour(i),tour(i+1))=1;
    A(tour(i+1),tour(i))=1;
end
A(tour(1),tour(n))=1;
A(tour(n),tour(1))=1;
end
```

## A.2 Nearest-neighbor

```
%Mariëlle Kruithof
%Nearest neighbor algorithm
%Input: symmetric distance matrix D.
%Output: adjacency matrix of a tour, tour and tour length.

function [tour,lengte,A]=nearest(D)
kleinstetourlength=zeros(length(D),1);
kleinstetour=zeros(length(D),length(D));
DD=D;               %To keep the original distance matrix.

for j=1:length(D)
    v=j;
    D=DD;
```

```
G=max(max(D));      %Take the maximum number of the matrix D.
vv=v;               %To keep the starting vertex.
[n,n]=size(D);      %The dimension of the matrix.
r=zeros(1,n);       %Vector to keep the distances.
k=zeros(1,n);       %Vector to keep the order of the tour.
k(1)=vv;            %Start with the start vertex.
    for i=1:n       %Place higher numbers on the diagonal.
        D(i,i)=G+1;
    end

%Start the algorithm.
for i=1:n-1
[a,b]=min(D(v,:));  %The smallest integer in the row corresponding to v.
r(i)=a;
D(v,:)=G+1;         %Replace the values of the visited vertex with high
D(:,v)=G+1;         %numbers.
k(i+1)=b;
v=b;
end
r(n)=DD(vv,v);      %Add the closing vertex to the tour.
%k(n+1)=vv;

lengte=sum(r);

kleinstetourlength(vv,:)=lengte;
kleinstetour(vv,:)=k;
end
[aa,bb]=min(kleinstetourlength);
lengte=aa;
tour=kleinstetour(bb,:);
A=adjacency(tour);          %Call the function adjacency.
end
```

## A.3 Farthest insertion

```
%Mariëlle Kruithof
%Farthest insertion algorithm
%Input: symmetric distance matrix D.
%Output: adjacency matrix of a tour, tour and tour length.

function [tour,tourlengte,A]=farthest(D)
[n,~]=size(D);                  %The dimension of the matrix.

for i=1:n
    for j=1:n
        if D(i,j)==max(max(D))  %Search for the highest-cost edge.
```

```
            g(1,1)=i;           %First node.
            g(2,1)=j;           %Second node.
        end
    end
end

for i=1:n                       %Set the diagonal at infinity.
    D(i,i)=inf;
end

totalnodes=(1:n);               %Make a list of the total number of nodes.
k=n-2;

while k~=0
nodes=totalnodes;

for i=1:length(g)
    nodes(g(i))=0;
end
nodes(nodes==0)=[];             %Remove all zeros.

l=zeros(length(g),1);
p=zeros(length(nodes),1);

for i=1:length(nodes)
    s=1;
    for j=1:length(g)
        l(s)=D(nodes(i),g(j));
        s=s+1;
    end
    p(i)=min(l);
end
[a,b]=max(p);
newnode=nodes(b);               %The new node.

%The highest minimal value is reached with this neighbor.
buur=find(D(newnode,:)==a);

%Decide where to place the new node.
plek=zeros(length(g),1);
for i=1:length(g)-1
    plek(i)=D(g(i),newnode)+D(newnode,g(i+1))-D(g(i),g(i+1));
end
plek(length(g))=D(g(length(g)),newnode)+D(newnode,g(1))-D(g(1),g(length(g)));

%Decide at which position the tour increases least.
[aa,bb]=min(plek);
```

```
for i=length(g):-1:bb+1
      g(i+1)=g(i);
end
g(bb+1)=newnode;                %Place the new node in the tour.

k=k-1;
end

%Getting tour length.
tourlengte=0;
for i=1:n-1
    lengte=D(g(i),g(i+1));
    tourlengte=tourlengte+lengte;
end
tourlengte=tourlengte+D(g(1),g(n));
tour=g';
A=adjacency(tour);              %Call the function adjacency.
end
```

## A.4 Biased random sampling

```
%Mariëlle Kruithof
%Biased random sampling
%Input: distance matrix D.
%Output: adjacency matrix of a tour, tour and tour length.

function [A, tour,tourlengte]=biased(D)
[n,~]=size(D);                  %The dimension of the matrix.
kleinstetourlength=zeros(length(D),1);
kleinstetour=zeros(length(D),length(D));
uv=rand(1,n);    %make a uniformly distributed vector uv.

for j=1:length(D)
    q=j;

g(1,1)=q;                       %The tour starts with the starting vertex q.

totalnodes=(1:n);               %Make a list of the total number of nodes.
k=n-2;

while k~=0

nodes=totalnodes;
for i=1:length(g)
    nodes(g(i))=0;
```

```
end
nodes(nodes==0)=[] ;            %Remove all zeros.
w=zeros(1,length(nodes));       %Make a zero row.
s=g(length(g));                 %The last inserted vertex in the tour.
c=zeros(1,length(nodes));
for i=1:length(nodes);          %Make a cost function of unused vertices.
   c(1,i)=D(s,nodes(i));
end

for i=1:length(nodes);          %Compute the weights of the unused vertices.
    w(1,i)=(1/c(i)^-1)/(1/sum(c));
end
sumw=sum(w);
for i=1:length(w)               %Normalize w.
    w(i)=w(i)/sumw;
end
u=uv(k);                         %Take a number of the vector uv.

ww=zeros(1,length(w));
for i=1:length(w)
    ww(i)=sum(w(1:i));
end

for i=1:length(ww)
    if ww(i)>u
        break
    end
end
if i==1                         %If the formula does not hold at all.
    volgende=nodes(i);
else
volgende=nodes(i-1);
end
g(1,length(g)+1)=volgende;
k=k-1;
end

%Insert the last node in the tour.
nodes=totalnodes;
for i=1:length(g)
    nodes(g(i))=0;
end
nodes(nodes==0)=[];             %Remove all zeros.
g(n)=nodes(1);

%Getting the length of the tour.
tourlengte=0;
```

```
for i=1:n-1
    lengte=D(g(i),g(i+1));
    tourlengte=tourlengte+lengte;
end
tourlengte=tourlengte+D(g(1),g(n));

%Make a matrix with all possibilities.
kleinstetourlength(q,:)=tourlengte;
kleinstetour(q,:)=g;
clear g
clear c
clear w
end
[aa,bb]=min(kleinstetourlength);
tourlengte=aa;
tour=kleinstetour(bb,:);
A=adjacency(tour);              %Call the function adjacency.
end
```

# Appendix B

# Van der Veen

```
%Mariëlle Kruithof
%Van der Veen
%Input: circulant symmetric distance matrix B.
%Output: lower bound VDV for the SCTSP.

function VDV=Vanderveen(B)

n=length(B);
if n == 2*round(n/2)            %n even
    d = n/2;
else                            %n odd
    d =  (n-1)/2;
end

r=zeros(d+1,2);                 %The entries of D.

for i=1:d+1
    r(i,1)=B(1,i);
    r(i,2)=i-1;                 %Extract the indices of r.
end

sort=zeros(d+1,2);
sort=sortrows(r);               %Sort the elements of r in ascending order.
phi=zeros(d+1,1);
phi=sort(:,2);

GCDD=ones(d+1,1);
gcd1=gcd(n,phi(2));         %Take the gcd of n and the first element of phi.
GCDD(1,1)=n;
GCDD(2,1)=gcd1;
    if gcd1==1
       l=1;
   else
```

```
     for i=3:d+1                %Calculate l.
         gcd1=gcd(gcd1,phi(i));
         GCDD(i,1)=gcd1;
         if gcd1==1
             l=i-1;
             break
         end
     end
    end
sumGCD=0;
for i=2:l+1                     %Summation to obtain de lowerbound.
    sGCD=(GCDD(i-1)-GCDD(i))*r(phi(i)+1);
    sumGCD=sumGCD+sGCD;
end
VDV=sumGCD+r(phi(l+1)+1);       %Add the last edge to make a cycle.

end
```

# Appendix C

# Held-Karp

## C.1 Maximum flow

```
%Maximum flow
%Input: source and solution matrix c.
%Output: flow and p.

function [flow,p]=maxFlowAll(source,c)

% zero flow ---  residual capacity = capcacity
[n,~]=size(c);
p=zeros(1,n);
v=zeros(1,n);
p(n)=1;
flow=0;
while p(n) ~= 0
    for i=1:n
        p(i)=0;
    end;
    v(1)=source; level=1;
    p(source)=n;
    while ((level~=0) && (p(n)==0))
        x=v(level) ;
        level=level-1;
        for j=1:n
            if (c(x,j)~=0) && (p(j)==0)
                p(j)=x ;
                level=level+1 ;
                v(level)=j;
            end
        end
     end
    if p(n)~=0
         k=n;
```

```
        rmin=999999999999;
        i=0;
        while i~=source
         i=p(k); j=k;
         if rmin>c(i,j)
            rmin=c(i,j);
         end
         k=p(k);
        end
        flow=flow+rmin;
        k=n;
        i=0;
        while i~=source
        i=p(k); j=k;
        c(i,j)=c(i,j)-rmin;
        c(j,i)=c(j,i)+rmin;
        k=p(k);
        end
    end
end
```

## C.2 Held-Karp bound

```
%Held-Karp bound
%Input: symmetric distance matrix B.
%Output: lower bound, solution matrix and number of cuts.

function [obj,matX,nrCuts]=HeldKarpCD(B)
tic                             %To measure the running time.
n = length(B);

e = ones(n,1);

X = sdpvar(n,n);                %Create the empty solution matrix X.
                                %Insert the constraints.
F = set(  diag(X) == zeros(n,1) );
F = F + set( X*e == 2*e );
F = F + set( X(:) >= 0 );
F = F + set( X(:) <= 1 );

obj = trace(B*X)/2;

%Solve the problem ignoring the subtour constraints:
diagnostics=solvesdp(F,obj,sdpsettings('showprogress',1,'solver','sedumi'))

%Take the subtour constraints into account by computing the max flow.
```

```
matX=double(X);
matX(((abs(matX)<1e-5))) = 0;
for s=1:n-1
[flow,p]=maxFlowAll(s,matX);    %Call the function maxFlowAll.
if flow <1.99999
    break;
end;
end;

if flow >1.99999
  [flow,p]=maxFlowAll(1,matX);
end;

nrCuts=0;                       %Start with cutting the polytope.
while (flow < 1.99999)
    k=0; q=0;
    for i=1:n
        if p(i)~= 0
            k=k+1;
            cut(k)=i;
        else
            q=q+1;
            cutcut(q)=i;
        end;
    end;

    E=zeros(n);
    for i=1:k
        for j=1:q
                E(cut(i),cutcut(j))=1;
        end;
    end;
    clear cut;
    clear cutcut;
    F = F + set( trace(E*X) >= 2 );
    nrCuts=nrCuts+1
diagnostics=solvesdp(F,obj,sdpsettings('showprogress',1,'solver','sedumi'))
    matX=double(X);
    matX(((abs(matX)<1e-5))) = 0;

    for s=1:n-1
    [flow,p]=maxFlowAll(s,matX);
    if flow <1.99999
      break;
    end;
    end;
```

```
    if flow >1.99999
      [flow,p]=maxFlowAll(1,matX);
    end;

end;
    toc
```

# Appendix D

# Semidefinite programming

```
%Semidefinite programming
%Input: symmetric disctance matrix B.
%Output: fractional adjacency matrix that approximates the adjacency
%        matrix of the minimum length tour.



function solX=qaptlsp7(B)

n = length(B);
if n == 2*round(n/2)            % n even
    d = n/2;
else                            % n odd
    d =  (n-1)/2;
end

ss1 = 0;                        %Set the constants.
I = eye(n);
J = ones(n,n);


F = set([]);                    %Set the constraints.

for k=1:d,
    X{k} = sdpvar(n);
    F = F + set( X{k}(:)>=0);
    ss1 = ss1 + X{k};
end;

F = F + set( ss1 == J-I );

for i=1:d,
    pom = 0;
```

```
    for k=1:d,
        coss = cos(2*pi*i*k/n);
        pom = pom + coss*X{k};
    end;
    F = F + set( (I + pom) >=0 );
end;

obj = 0.5*trace(B*X{1});

%Call the solver Sedumi.
diagnostics=solvesdp(F,obj,sdpsettings('showprogress',1,'solver','Sedumi'));
solXX=X(1);
solX=double(solXX{1,1});
solX(((abs(solX)<1e-5))) = 0;
end
```

# Appendix E

# Tour construction

```
%Mariëlle Kruithof
%Rounding
%Input: solution matrix of the LP/SDP-relaxation X and distance matrix D.
%Output: constructed tour and tour length.

function [tour,tourlengte]=rounding(X,D)
n=length(X);            %get the dimension of D.
tour=zeros(1,n);        %In order to remember the tour.
dubbeltour=(1:n);       %In order to see what's used in the tour.
L=find(X(1,:)==1);      %Display the places that are equal to one.
tour(1,1)=1;
if size(L,2)==1         %If there is one entry equal to one.
    X(1,L)=-1;
    [~,b]=max(X(1,:));  %Find the second maximum value.
    X(1,:)=0;           %Put everything zero.
    X(1,b)=1;           %Put a one for the first edge.
    X(1,L)=1;           %Put a one for the second edge.
    X(:,1)=0;           %To make sure the column is zero.
    X(b,1)=1;
    X(L,1)=1;
    volgende=L;         %Remember the next vertex.
    laatste=b;
elseif size(L,2)==2;    %If there are two entries equal to one.
       X(:,1)=0;
    X(L(1),1)=1;
    X(L(2),1)=1;
    volgende=L(1);
    laatste=L(2);
else                    %If there are no entries equal to one.
    [~,b]=max(X(1,:));  %Find the maximum value.
    X(1,b)=-1;
    [~,d]=max(X(1,:));  %Find the second maximum value.
    X(1,:)=0;
```

```
    X(1,b)=1;
    X(1,d)=1;
    X(:,1)=0;
    X(b,1)=1;
    X(d,1)=1;
    volgende=b;
    laatste=d;
end
tour(1,2)=volgende;     %The first edge in the tour.
tour(1,n)=laatste;      %The closing edge in the tour.
vorige=1;
eruit=find(dubbeltour==vorige);
dubbeltour(eruit)=[];
eruit2=find(dubbeltour==laatste);
dubbeltour(eruit2)=[];


%Go to the next vertex as the current vertex.
for i=1:n-3
      X(volgende,vorige)=-1;
      [a,bb]=max(X(volgende,dubbeltour));
      b=dubbeltour(bb);
       if a==0               %If there are only zeros.
          b=dubbeltour(1);   %Choose the first unused vertex.
       end

      X(volgende,:)=0;
      X(volgende,vorige)=1;
      X(volgende,b)=1;
      X(:,volgende)=0;
      X(vorige,volgende)=1;
      X(b,volgende)=1;
       tour(1,i+2)=b;
       eruit=find(dubbeltour==volgende);
       dubbeltour(eruit)=[];         %Remove the already used vertices.
       eruit2=find(dubbeltour==b);
       dubbeltour(eruit2)=[];        %Remove the next current vertex.
       vorige=volgende;              %Remember the current vertex.
      volgende=b;             %The next vertex becomes the current vertex.

end
%Set the closing vertex.
X(volgende,:)=0;       %In order to obtain an adjacency matrix.
X(volgende,vorige)=1;
X(volgende,laatste)=1;
X(laatste,:)=0;
X(laatste,1)=1;
```

```
X(laatste,volgende)=1;

tourlengte=0;                  %Getting the length of the tour.
for i=1:n-1
    lengte=D(tour(i),tour(i+1));
    tourlengte=tourlengte+lengte;
end
tourlengte=tourlengte+D(tour(n),tour(1));
```

# Bibliography


[1] R.K. Ahuja, J.B. Orlin, and T.L. Magnanti. *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall, Englewood Cliffs, New Jersey, 1993.

[2] W.J. Cook, W.H. Cunningham, W.R. Pulleyblank, and A. Schrijver. *Combinatorial Optimization*. New York, USA: John Wiley and Sons, 1998.

[3] L.R. Ford, Jr. and D.R. Fulkerson. Maximal flow through a network. *Canadian Journal of Mathematics*, (8):399–404, 1956.

[4] L.R. Ford, Jr. and D.R. Fulkerson. *Flows in Networks*. Princeton University Press, Princeton, New Jersey, 1962.

[5] R.S. Garfinkel. Minimizing wallpaper waste, part 1: A class of traveling salesman problems. *Operations Research*, 25(5):741–751, 1977.

[6] F. Glover. Tabu search- a tutorial. *Interfaces*, 20(4):74–94, 1990.

[7] M. Held and R.M. Karp. The traveling salesman problem and minimum spanning trees. *Operations Research*, 18(6):1138–1162, 1970.

[8] K. Helsgaun. An effective implementation of the lin-kernighan traveling salesman heuristic. *European Journal of Operational Research*, 126:106–130, 1999.

[9] R.A. Horn and C.R. Johnson. *Matrix Analysis*. Cambridge, MA, USA: Cambridge University Press, 1990.

[10] R. Jonker and T. Volgenant. Transforming asymmetric into symmetric traveling salesman problems. *Operations Research letters*, 2(4):161–163, 1983.

[11] H. Karp. Complexity of computer computations. In R.E. Miller and J.W. Thatcher, editors, *Reducibility among combinatorial problems*. New York, 1972.

[12] T. Klastorin, P. Gray, et al. *Tutorials in Operations Research: OR Tools and Applications: Glimpses of Future Technologies*. Institute for Operations Research and the Management Sciences (INFORMS), 2007.

[13] E. de Klerk. Exploiting special structure in semidefinite programming: A survey of theory and applications. *European Journal of Operational Research*, 201(1):1–10, 2010.

[14] E. de Klerk and C. Dobre. A comparison of lower bounds for the symmetric circulant traveling salesman problem. *Discrete Applied Mathematics*, 159(16):1815–1826, 2011.

[15] E. de Klerk, D.V. Pasechnik, and R. Sotirov. On semidefinite programming relaxations of the traveling salesman problem. *SIAM Journal on Optimization*, 19(4):1559–1573, 2008.

[16] G. Laporte. The traveling salesman problem: An overview of exact and approximate algorithms. *European Journal of Operational Research*, 59:231–247, 1991.

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

[18] M.L. Overton. Linear programming. *Draft for Encyclopedia Americana*, 1997.

[19] H.D. Ratliff and A.S. Rosenthal. Order-picking in a rectangular warehouse: A solvable case of the traveling salesman problem. *Operations Research*, 31(3):507–521, 1983.

[20] G. Righini. The largest insertion algorithm for the traveling salesman problem. *Note del Polo-Ricerca*, 29, 2000.

[21] A. Schrijver. *Combinatorial Optimization*, volume B - Polyhedra and Efficiency. Springer, 2003.

[22] A. Schrijver. *A course in combinatorial optimization.* Department of Mathematics, University of Amsterdam, 2006.

[23] L. Vanderberghe and S. Boyd. Semidefinite programming. *SIAM Review*, 38(1):49–95, 1996.

[24] J.A.A. van der Veen. *Solvable Cases of TSP with Various Objective Functions.* PhD thesis, Groningen University, The Netherlands, 1992.

[25] D.P. Williamson. Analysis of the held-karp heuristic for the traveling salesman problem. Master's thesis, Massachusetts Institute of Technology, 1989.