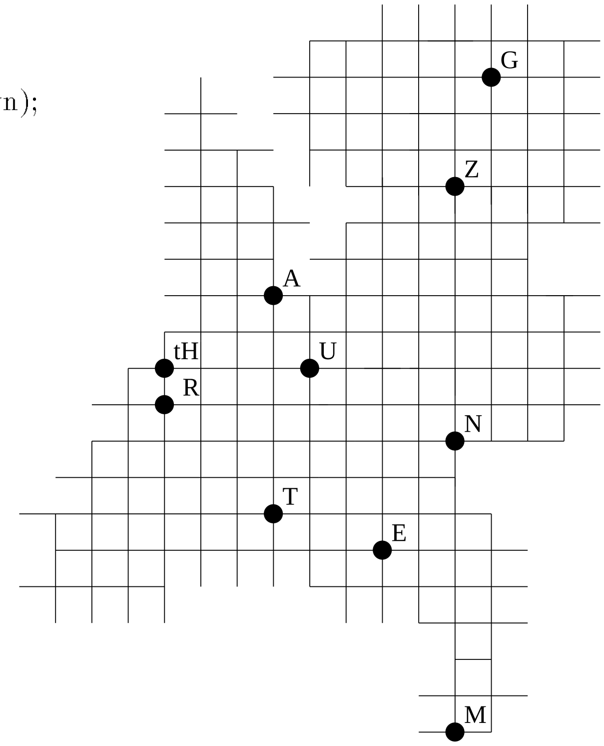
Chapter 5. The traveling salesman problem

Section 5.1. Problem definition

The traveling salesman problem (TSP).

A big company in the Netherlands sends one of its salesmen to the clients in the major cities in the country. A list of these cities together with their distances is given below. Starting and finishing in his hometown, the salesman has to visit these cities all in one day. Since he wants to return home as soon as possible, he must visit the cities in such an order that the total traveling distance is minimal (where we assume that the traveling time depends on the distance only).

Cities:
Amsterdam;
Eindhoven (home town);
Groningen;
the Hague;
Maastricht;
Nijmegen;
Rotterdam;
Tilburg;
Utrecht;
Zwolle.



The distance matrix is the following.

	A	E	G	tH	M	N	R	T	U	Z
A		125	184	57	207	103	77	121	40	103
E			235	127	125	62	101	37	91	144
G				236	309	192	242	233	184	94
tH					217	124	26	102	61	151
M						120	191	119	176	218
N							96	93	62	105
R								76	58	148
T									78	166
U										90
Z										

We have assumed the distance-matrix to be symmetric; therefore, the part above the main diagonal contains all necessary information.

The problem can be modeled as a minimization problem in graphs as follows. We are given a complete graph $G = (V, E)$ and a distance function $d : E \rightarrow R^+$. The problem is to find a Hamiltonian circuit of minimum total length that contains each vertex of V exactly once. Such a circuit is described by a permutation $\pi = (\pi(1), \ldots, \pi(n))$ of the vertices, where $\pi(i)$ is the vertex that is on the i-th position assuming that we start with vertex $\pi(1)$. The length of the circuit described by π is therefore

$$\sum_{i=1}^{n} d_{\pi(i),\pi(i+1)} \quad (\text{where } \pi(n+1) = \pi(1)) \; .$$

In the usual terminology for traveling salesman problems, the vertices are called cities and Hamiltonian circuits are called tours.

Variants.
The problem above is called the *symmetric* TSP, because the distance between each pair of cities is independent of the direction in which the salesman travels. This restriction is not always valid; we then obtain the following, more general, form of the TSP.

The asymmetric TSP.
Given a complete digraph $G = (V, A)$ and a length function $d : A \rightarrow R^+$, we are asked to find a directed Hamiltonian circuit, that is, a directed circuit that contains each vertex exactly once, of minimum length.

Both the symmetric and the asymmetric TSP are defined on a complete (di)graph. This is not a serious restriction, however, since we can always add arcs or edges with a sufficiently high cost; sufficiently should be read as: if there exists a tour in the original graph, then none of the extra arcs or edges will be part of the tour of minimum length.

The Hamiltonian path problem.
If the salesman does not have to return home, then we get the Hamiltonian path problem: Given a complete graph $G = (V, A)$ and a length function $d : A \rightarrow R^+$, we are asked to find a Hamiltonian path, that is, a path that contains each vertex exactly once, of minimum length. There are three variants of this problem, depending on the number of endpoints that are fixed.

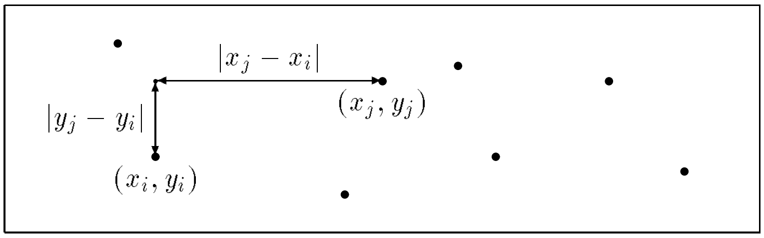
The Hamiltonian path problem is easily converted to a standard TSP problem by adding a dummy vertex with appropriate distance to all other vertices, where the specification of appropriate depends on the number of endpoints that have been fixed. If no endpoints have been fixed, then we are indifferent to which vertices are connected to the dummy vertex (and hence will star as the endpoints of the path). If a vertex has been specified as endpoint, then we have to take care that this vertex is connected to the dummy vertex in the solution of the TSP; this is arranged by setting the distance from this point to the dummy vertex equal to zero, whereas all vertices that are not specified as endpoint get a huge distance to the dummy vertex. On the other hand, it is not hard to convert a TSP problem into a minimum length Hamiltonian path problem.

There are no polynomial algorithms known for the TSP and its variants. It can be shown that the partition problem is a special case, but that is beyond the scope of this manuscript.

Section 5.2. Applications

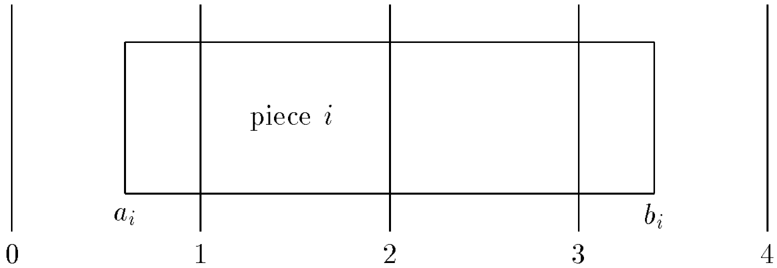
A natural application of the TSP is the Euclidean TSP, in which the cities are given by points in the plane; the distances between pairs of cities are the Euclidean distances. The following application is of a similar type.

Printed circuit boards.
In the process of manufacturing electronic circuit boards, holes have to be drilled in prints of rectangular shape; these holes have to be pairwise connected by conductive material. The head of the drilling equipment can only be moved horizontally and vertically from one hole to another. Usually, an enormous amount of holes has to be drilled in a board. Since moving the head of the drill is a time consuming operation, the total distance that the head has to be moved should be minimized. Moreover, after the drilling process has been completed, the head must return to its starting position. Hence, this problem is equivalent to the problem of finding a Hamiltonian circuit of minimum length in a graph in which the vertices correspond to the holes and the distance between two vertices i and j with coordinates (x_i, y_i) and (x_j, y_j) amounts to $|x_i - x_j| + |y_i - y_j|$.



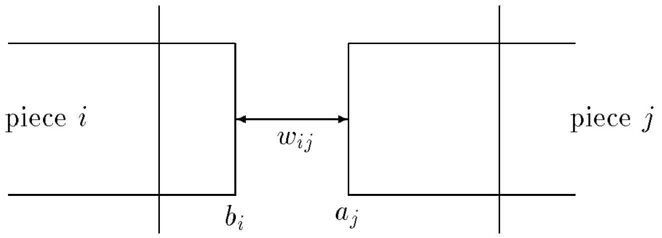
This problem resembles the Euclidean TSP. The only difference between the two problems is the definition of the distances between pairs of points in the plane. For the printed circuit boards the 1-norm is taken, whereas the Euclidean TSP uses the 2-norm.

Wallpaper cutting.
Suppose we are to cut $n - 1$ sheets of wallpaper from a very long roll of wallpaper to cover a wall in a house. The roll contains a printed pattern that repeats itself every meter. Each piece i ($i = 1, \ldots, n$) starts at position a_i and finishes at position b_i with respect to the point where the pattern starts; these positions depend on the pieces, that is, they may be different for distinct pieces, because of the presence of windows, doors, etc., in the wall.

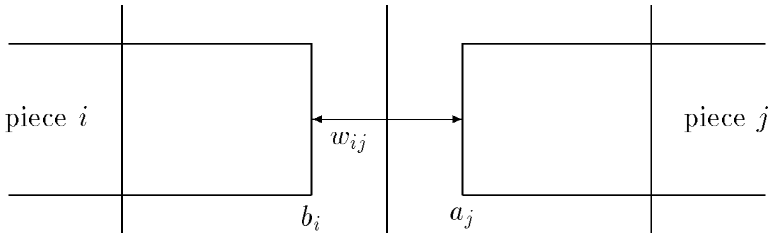


Now suppose that piece j is cut off the roll immediately after piece i. Then the wastage w_{ij} between the pieces depends on whether $a_i \geq b_j$ or not.

Case 1: Let $a_j \geq b_i$.



Hence, the wastage amounts to $w_{ij} = a_j - b_i$.
Case 2: Let $a_j < b_i$.



Hence, the wastage amounts to $w_{ij} = 1 + a_j - b_i$.
We have now characterized the wastage between the pieces. Suppose the roll begins at the zero point on the pattern and after cutting our $n-1$ pieces we have to make an additional cut to let the roll start at the zero point on the pattern again. To take into account the wastage incurred by the first and the last cut, we add a dummy piece n with $a_n = b_n = 0$. This turns the problem into an asymmetric TSP problem with the distance matrix is $W = (w_{ij})$. Surprisingly enough this special case of the TSP can be solved in polynomial time.

Clustering a data array.
Suppose we are given a data array in the form of an $m \times n$ matrix $A = (a_{ij})$ that consists of elements that are either 0 or 1: the entry a_{ij} gets the value 1 if a relation between row i and column j exists. If there is no relation between the row and the column, then $a_{ij} = 0$. We are interested in grouping rows and columns together in such a way that they show similar relations.

For example, consider a number of m marketing techniques and n products. If a marketing technique i works out successfully on a product j, then $a_{ij} = 1$; otherwise, $a_{ij} = 0$. Similar marketing techniques are supposed to be successful on similar products. Therefore, grouping the techniques and the products gives insight in the relations of the marketing techniques and the products.

Consider the following two matrices. The second matrix is constructed from the first one by permuting rows 2 and 5 and columns 2 and 5.

$$\begin{array}{|ccccc|}\hline 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ \hline \end{array} \longrightarrow \begin{array}{|ccccc|}\hline 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ \hline \end{array}$$

Intuitively, the matrix on the right gives a better idea of which rows and columns have a relation. To formalize this, we introduce *the measure of effectiveness* for each element a_{ij} of the matrix; it is defined as the product of this element with its (at most) four direct neighbors; these are, the upper and lower and the left and right neighbors, if they exist. To ensure their existence, we add artificial rows 0 and $m+1$ and columns 0 and $n+1$ that contain zeroes only to the matrix A. The total measure of effectiveness (TME) is computed by summing the measure of effectiveness over all elements of the matrix except for the artificial rows and columns:

$$TME = \sum_{i=1}^{m} \sum_{j=1}^{n} a_{ij}(a_{i-1,j} + a_{i+1,j} + a_{i,j-1} + a_{i,j+1}).$$

The TME's of the two matrices in the example are 0 and 32, respectively. For arbitrary permutations ρ and σ of the rows $1, \ldots, m$ and columns $1, \ldots, n$, the TME becomes

$$TME = \sum_{i=1}^{m} \sum_{j=1}^{m} a_{\rho(i)\sigma(j)}(a_{\rho(i)\sigma(j-1)} + a_{\rho(i)\sigma(j+1)} + a_{\rho(i-1)\sigma(j)} + a_{\rho(i+1)\sigma(j)})$$

$$= \sum_{i=1}^{m} \sum_{j=1}^{n} [a_{\rho(i)\sigma(j)} a_{\rho(i)\sigma(j-1)} + a_{\rho(i)\sigma(j)} a_{\rho(i)\sigma(j+1)}] \quad (1)$$

$$+ \sum_{i=1}^{m} \sum_{j=1}^{n} [a_{\rho(i)\sigma(j)} a_{\rho(i-1)\sigma(j)} + a_{\rho(i)\sigma(j)} a_{\rho(i+1)\sigma(j)}]. \quad (2)$$

The interpretation of (1) is the following. For the row placed on the i-th position, that is, the row denoted by $\rho(i)$, each element is multiplied with its predecessor and its successor with respect to σ. Rewriting this formula yields:

$$\sum_{i=1}^{m} \sum_{j=1}^{n} [a_{\rho(i)\sigma(j)} a_{\rho(i)\sigma(j-1)} + a_{\rho(i)\sigma(j)} a_{\rho(i)\sigma(j+1)}] =$$

$$\sum_{k=1}^{m} \sum_{j=1}^{n} [a_{k\sigma(j)} a_{k\sigma(j-1)} + a_{k\sigma(j)} a_{k\sigma(j+1)}] =$$

$$\sum_{k=1}^{m} \sum_{j=1}^{n} a_{k\sigma(j)} a_{k\sigma(j-1)} + \sum_{k=1}^{m} \sum_{j=1}^{n} a_{k\sigma(j)} a_{k\sigma(j+1)} =$$

$$\sum_{k=1}^{m}\sum_{j=0}^{n-1} a_{k\sigma(j+1)}a_{k\sigma(j)} + \sum_{k=1}^{m}\sum_{j=1}^{n} a_{k\sigma(j)}a_{k\sigma(j+1)} \qquad =$$

$$2\sum_{k=1}^{m}\sum_{j=1}^{n} a_{k\sigma(j)}a_{k\sigma(j+1)} = 2\sum_{j=1}^{n}\left(\sum_{k=1}^{m} a_{k\sigma(j)}a_{k\sigma(j+1)}\right) \qquad =$$

$$2\sum_{j=1}^{n} a_{*\sigma(j)}a_{*\sigma(j+1)},$$

where the first equality holds since the sum is taken over all elements $\rho(i), (i = 1, \ldots, n)$ and the fourth since $a_{k\sigma(0)} = a_{k\sigma(n+1)} = 0$; $a_{*\sigma(j)}a_{*\sigma(j+1)}$ is defined as the *inner product* of the columns $\sigma(j)$ and $\sigma(j+1)$. Hence, we have to minimize the sum over the inner products of columns that are neighbors with respect to the permutation σ. If we define the distance d_{rs} between the columns r and s $(0 \leq r < s \leq n+1)$ as their inner product, which we multiply by -1 to create a minimization problem, that is,

$$d_{rs} = -2a_{*r}a_{*s} = -2\sum_{k=1}^{m} a_{kr}a_{ks}$$

then we obtain the problem of finding an undirected Hamiltonian path of minimum length between the columns 0 and $n+1$.

Similarly, (2) can be rewritten as the summation over the inner products of the rows. An analogous analysis leads to the problem of finding a minimum length Hamiltonian path between the rows 0 and $m+1$, where the distances are defined by

$$d_{rs} = -2a_{r*}a_{s*} = -2\sum_{j=1}^{n} a_{rj}a_{sj}.$$

We see that the problem of clustering a data array can be decomposed into two Hamiltonion path problems, one defined on the rows and one defined on the columns.

Section 5.3. Integer linear programming formulations
The asymmetric TSP.
Consider the asymmetric TSP with distance matrix d_{ij}. We will show how this problem can be modeled as an integer linear programming problem. To that end, we introduce decision variables x_{ij} $(1 \leq i, j \leq n)$, the outcome of which is interpreted as follows:

$$x_{ij} = \begin{cases} 1, & \text{if the arc from city } i \text{ to city } j \text{ is included in the tour} \\ 0, & \text{otherwise .} \end{cases}$$

Clearly, we need the constraints

$$\forall_{j\in V} \; \sum_{i=1}^{n} x_{ij} = 1, \tag{1}$$

and

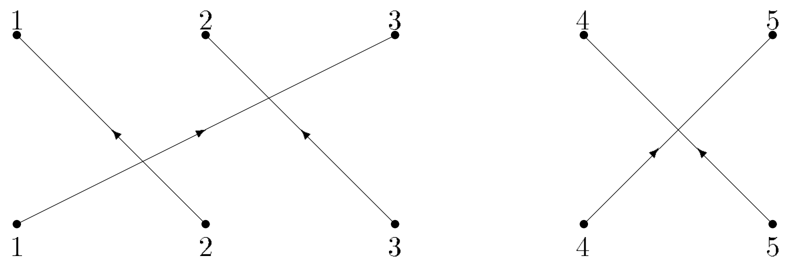
$$\forall_{i\in V} \sum_{j=1}^{n} x_{ij} = 1, \tag{2}$$

since we enter and leave each city exactly once. However, this is not enough. The problem with constraints (1) and (2) and the integrality constraints is a relaxation of the asymmetric TSP, since it allows for solutions that do not correspond to tours. This is illustrated by the following example.

Example.

	1	2	3	4	5
1		1	1	4	4
2	1		1	4	4
3	2	1		4	4
4	3	3	3		1
5	3	3	3	1	

Since the outdegrees and the indegrees of all vertices have to be equal to one, the problem with the integrality constraints and (1) and (2) is a minimum weight bipartite matching problem, with the rows corresponding to a set of vertices V_1 and the columns corresponding to a set of vertices V_2. The optimal solution of the example is simply found. It corresponds to two subtours, one containing the vertices $\{1, 2, 3\}$ and one containing $\{4, 5\}$.



We need constraints to prohibit subtours from occurring. The following restrictions are sufficient to limit the set of feasible solutions to tours:

$$\forall_{S\subset\{1,\ldots,n\},\emptyset\neq S\neq V} \sum_{i,j\in S} x_{ij} \leq |S| - 1. \tag{3}$$

These restrictions force that each proper subset on k cities contains at most $k-1$ arcs; subtours clearly do not satisfy this condition. The following theorem shows that adding this extra set of constraints leaves us with a correct formulation.

Theorem 5.1.
Each subset of arcs that satisfies the restrictions (1), (2), (3), and the integrality conditions forms a directed Hamiltonian circuit, and each directed Hamiltonian circuit corresponds to a feasible solution of the above formulation.

Proof.
Consider any integral solution that satisfies (1), (2), and (3). Due to restrictions (1) and (2), we have that each vertex has indegree 1 and outdegree 1. Hence, the graph consists of one or more Eulerian subgraphs. Suppose that there are two or more components, that is, the solution is not connected. Take one of the connected components, say $S \neq V$; this S does not satisfy (3), contrary to our assumptions.

It is easy to check that a Hamiltonian circuit satisfies all constraints. □

Up till now, we have not specified the objective function. Since we are looking for a tour of minimal length, we want that our objective function resembles the total length of the tour. To that end, we choose the objective function, which we want to minimize, equal to

$$\sum_{i=1}^{n}\sum_{j=1}^{n} d_{ij}x_{ij}.$$

The symmetric TSP.
Consider the symmetric TSP with distance matrix d_e. This problem can be modeled as an integer linear programming problem as follows. For each edge $e \in E$, we introduce a decision variable x_e, the outcome of which is interpreted as follows:

$$x_e = \begin{cases} 1 & \text{if edge } e \text{ is in the tour ;} \\ 0 & \text{otherwise .} \end{cases}$$

The degree constraints are the following.

$$\forall_{i \in V} \sum_{e : e \ni i} x_e = 2. \tag{4}$$

The problem concerning the elimination of subtours arises in the symmetric case, too. The same solution, however, also works here, so we add the following constraints:

$$\forall_{S \subset \{1,\ldots,n\}, \emptyset \neq S \neq V} \sum_{e \subset S} x_e \leq |S| - 1. \tag{5}$$

Hence, each subgraph induced by S contains strictly less edges than vertices; we can prove in a similar fashion that the constraints (4) and (5) together with the integrality constraints yield a correct formulation. The objective function, which has to be minimized, is chosen equal to

$$\sum_{e \in E} d_e x_e.$$

Section 5.4. Approximation algorithms

Since no algorithm is known that solves the TSP to optimality in polynomial time, approximation algorithms have been developed to get quick and hopefully good solutions. These algorithms can roughly be divided into two classes: constructive algorithms and improvement algorithms. Constructive algorithms build a tour step by step, whereas improvement algorithms try to improve a given tour by exchanging edges in the tour with edges not in the tour. We provide some examples from both classes of algorithms.

Constructive approximation algorithms.

We discuss two algorithms for the symmetric TSP. In order to be able to say something about the relative performance of the algorithms, we assume that the length function satisfies the *triangle inequality*, that is, we have that $d_{ik} \leq d_{ij} + d_{jk}$ for all i, j, k in V.

The double tree algorithm (TT).

Algorithm TT consists of four phases. In the first three phases, we construct an Euler-cycle that we convert into a Hamilton-circuit in Phase 4.

Phase 1.

Construct a minimum spanning tree with respect to the distance function d.

Phase 2.

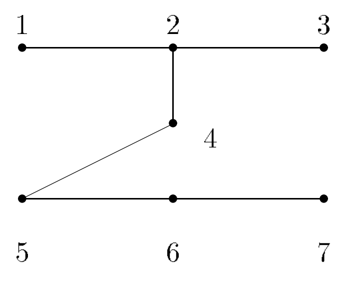
Double all edges in the tree to obtain an Eulerian graph.

Phase 3.

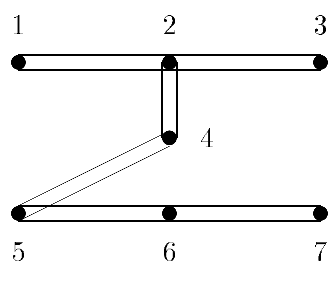
Determine an Euler-cycle in the Eulerian graph determined in Phase 2. Since the Eulerian graph is connected (it contains the minimum spanning tree as a subgraph), the cycle contains each vertex at least once.

Example

	1	2	3	4	5	6	7
1		1	2	2	3	4	5
2			1	1	3	3	4
3				2	3	2	3
4					1	2	3
5						1	2
6							1
7							



A minimum spanning tree



Doubling the tree

We now have to find an Euler-cycle in the graph depicted on the right-hand side. It is easily checked that 3212456765423 is an Euler-cycle.

Phase 4.
Convert the Euler-cycle into a Hamilton-circuit by applying *short-cuts*, as described below.

Shrinking a cycle by applying short-cuts.
Let the cities i, j, and k occur in the cycle in this order, that is, the edges $\{i, j\}$ and $\{j, k\}$ follow each other in the cycle. Replace these edges by the edge $\{i, k\}$, as is shown below.

This operation can be performed on any cycle. Its effect is that a new cycle is constructed with one edge less; in this cycle, there is one vertex that is visited one time less in comparison with the previous cycle. Due to the assumption that the length function satisfies the triangle inequality, we have that applying a short-cut does not increase the length of the cycle.

Lemma 5.2 Let $G = (V, E)$ be a complete graph, and let $d : E \rightarrow R^+$ be a distance function on the edges, which satisfies the triangle inequality. Let C be a cycle in this graph with total length $d(C)$. If C' is a cycle constructed from C by applying short-cuts, then we have that the total length of C' is no more than $d(C)$.

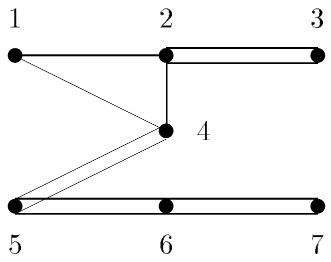
We apply a short-cut on each second appearance of a vertex j. Therefore, each vertex remains in the cycle at least once. After the short-cut has been applied to all second appearances of the vertices, the cycle contains each vertex exactly once, that is, we have constructed a Hamilton-circuit.

Example continued.

$321\underline{2}4567\underline{6542}3$

$\downarrow$ $124 \rightarrow 14$

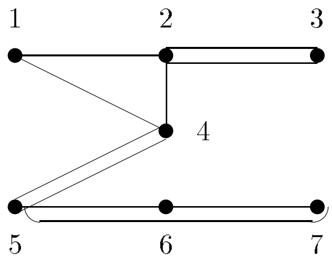
$3214567\underline{6542}3$



$3214567\underline{6542}3$

$\downarrow$ $765 \rightarrow 75$

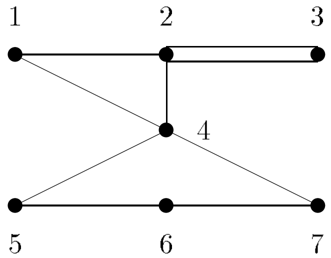
$3214567\underline{542}3$



$3214567\underline{542}3$

$\downarrow\ 754 \rightarrow 74$

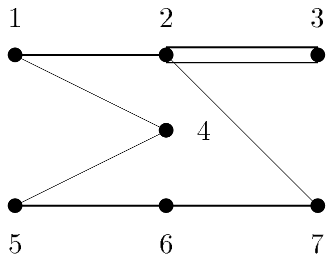
$3214567\underline{42}3$



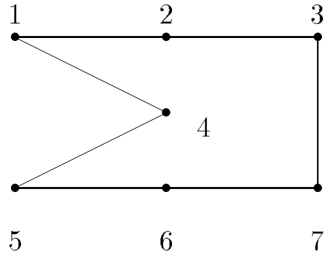
$3214567\underline{42}3$

$\downarrow\ 742 \rightarrow 72$

$3214567\underline{2}3$



32145673



The tour constructed by Algorithm TT

We now show that Algorithm TT will never produce a solution with length more than twice the length of an optimal solution. Hence, we say that Algorithm TT has *worst-case ratio* equal to 2.

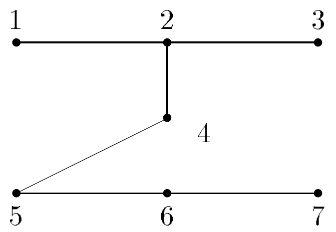
Let z_{opt} denote the length of an optimal tour, and let z_{TT} denote the length of the tour constructed by algorithm TT. Finally, let z_T denote the length of a minimum spanning tree. We first prove that $z_T \leq z_{\text{opt}}$; we then complete the proof by showing that $z_{TT} \leq 2 * z_T$.

1. Consider any optimal tour. If we delete an arbitrary edge from it, then we obtain a Hamiltonian path. Since the length of any edge is nonnegative because of the triangle inequality, the length of this Hamilton path amounts to no more than z_{opt}. Since a Hamiltonian path is a special case of a spanning tree, we have that its length amounts to at least z_T. Concluding, we have that $z_T \leq z_{\text{opt}}$.
2. The total length of the edges in the Eulerian graph that is constructed by doubling the minimum spanning tree is equal to $2 * z_T$. From Lemma 5.2, it follows that the length z_{TT} of the Hamiltonian circuit that is obtained by applying short-cuts amounts to no more than the length of the Eulerian cycle, which is equal to $2 * z_T$.

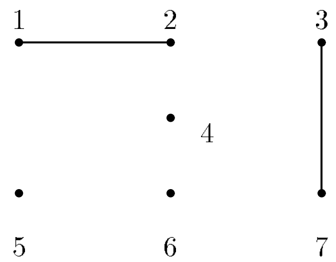
Combining both results, we get $z_{TT} \leq 2 * z_T \leq 2 * z_{\text{opt}}$.

The tree-matching algorithm (TM).
From the analysis above, it follows immediately that, if we want to improve on our worst-case ratio, then we should try to decrease the length of the Eulerian cycle. Recalling that a graph is Eulerian if and only if it is connected with even degree in each point, it seems obvious to start with a minimum spanning tree to make sure that the graph is connected. The only problem left is to take care of the vertices with odd degree; note that the number of vertices with odd degree is even. If we limit our attention to the set V_0 containing the vertices with odd degree, then we see that we can get even degree in each of these vertices by adding a perfect matching M on these vertices with minimum length. The Phases 1, 3, and 4 in Algorithm TM are identical to the corresponding phases in Algorithm TT.

Example continued.

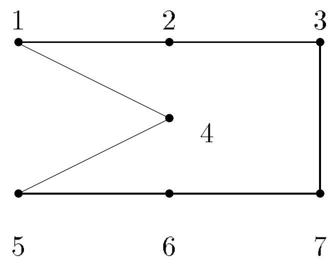


A minimum spanning tree



A perfect matching of minimum lenght

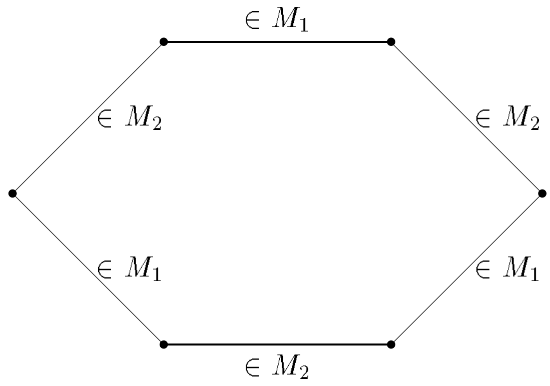
The combination of the tree and the matching depicted above yields an Eulerian graph that contains the Euler-cycle 1237654$\underline{2}$1. Applying the short-cut in which the edges $\{1,2\}$ and $\{2,4\}$ are replaced with the edge $\{1,4\}$ yields the Hamiltonian cycle that is depicted below. Note that for this example algorithm TT and algorithm TM find the same tour; this is a coincidence.



The tour constructed by Algorithm TM

This small change with respect to algorithm TT results in a better worst-case behaviour. For any instance of the TSP (with triangle-inequality), algorithm TM constructs a tour with length no more than 3/2 times the length of an optimal tour. To prove this, it suffices to show that the length z_M of the matching M is no more than $\frac{1}{2}$ times z_{opt}. Namely, then $z_{TM} \leq z_T + z_M \leq z_{\text{opt}} + \frac{1}{2} z_{\text{opt}} = \frac{3}{2} * z_{\text{opt}}$.

The proof makes use of the shrinking algorithm again. Consider any optimal tour with value z_{opt}. Apply short-cuts such that only the vertices in V_0 remain in the cycle; this yields a circuit C on the vertices in V_0 with length no more than z_{opt}. C can be partitioned into two complete matchings M_1 and M_2 on V_0 by 'walking' along the circuit and putting the first edge in M_1, the second edge in M_2, the third edge in M_1, etc., as is shown in the figure below. Note that the circuit contains an even number of edges, because $|V_0|$ is even.



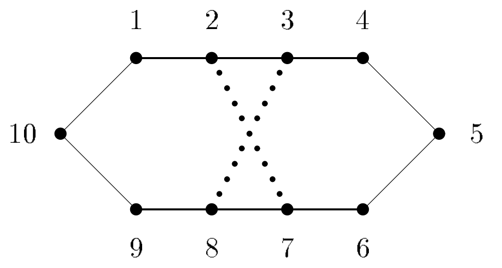
Since M_1 and M_2 are complete matchings on V_0, we have that $d(M) \leq d(M_1)$ and $d(M) \leq d(M_2)$. Hence, we have that $2 * d(M) \leq d(M_1) + d(M_2) = d(C) \leq z_{\text{opt}}$, which was to be proved.

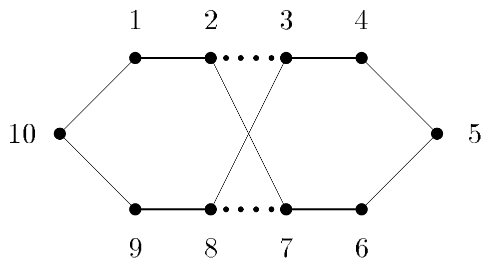
Note that for both algorithms our analysis of the worst-case ratio depends heavily on the assumption that the triangle inequality holds. It can be shown that, in case the triangle inequality fails to hold, the worst-case ratio is no longer fixed, that is, for any constant c an instance of the TSP can be constructed such that the length of the tour constructed by the algorithm is more than c times as long as the length of the optimal tour.

Improvement algorithms.
Suppose that we have a solution, that is, a tour, for a certain instance of the symmetric TSP. It may have been constructed by the previous algorithms or it may be just a random tour. An improvement algorithm tries to replace a subset of the edges of the tour by a new set of edges with smaller total length such that a new, shorter, tour results. We give four variants of improvement algorithms for the TSP.

2-edge-exchange (2-opt).
An iteration of 2-edge-exchange proceeds as follows. Take two edges that are not adjacent, say $\{i_1, j_1\}$ and $\{i_2, j_2\}$. Remove them from the tour. Let the remainder of the tour consist of the paths with endvertices i_1 and i_2 and endvertices j_1 and j_2, respectively. There are two ways to connect these two paths to a tour again: by reinserting the edges $\{i_1, j_1\}$ and $\{i_2, j_2\}$, which yields the old tour, and by inserting the edges $\{i_1, j_2\}$ and $\{i_2, j_1\}$. Choose the pair of edges with smaller cumulative length, that is, change the original tour only if you gain by doing so. Note that the insertion of the edges $\{i_1, i_2\}$ and $\{j_1, j_2\}$ creates a graph consisting of two subtours.

Example

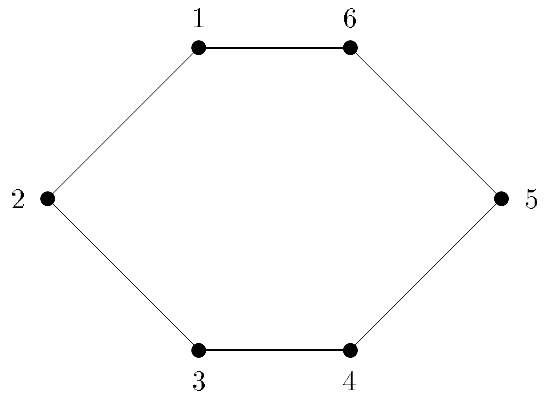




Remove the edges $\{2,3\}$ and $\{7,8\}$. To get a tour again, the edges $\{2,7\}$ and $\{3,8\}$ must be inserted. This is done only if the latter two edges have smaller cumulative length. Note that the addition of the edges $\{2,8\}$ and $\{3,7\}$ would split the tour into two subtours.

The 2-edge-exchange algorithm, or 2-opt for short, proceeds by performing edge exchanges until no pair of edges can be replaced by a better pair. The final tour is called 2-optimal. There is no guarantee at all that this algorithm ends after a polynomial number of steps. In practice, however, even if we start with a random tour, it stops after very few iterations. The typical number of iterations is quadratic in the number of cities n. Note that it takes $O(n^2)$ time to verify whether a tour is 2-optimal.

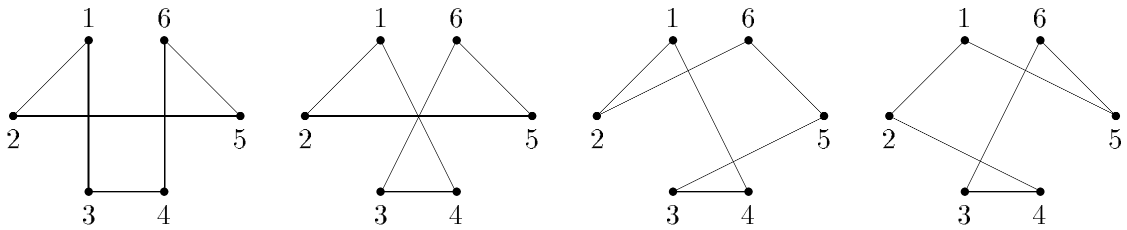
3-edge-exchange (3-opt).
The 2-opt procedure can be generalized by considering larger sets of edges, though the procedure becomes more and more involved then. In practice, we use exchanges of three edges at most at a time. An iteration is described as follows. Take three arbitrary edges, and try to replace them by three new edges, thus creating a new tour. In general, if none of the three edges are adjacent, then there are four candidate triples of edges that may replace the original triple. Consider the tour depicted below.



Remove the edges $\{1,6\},\{2,3\},\{4,5\}$. The four triples that can replace these edges are:

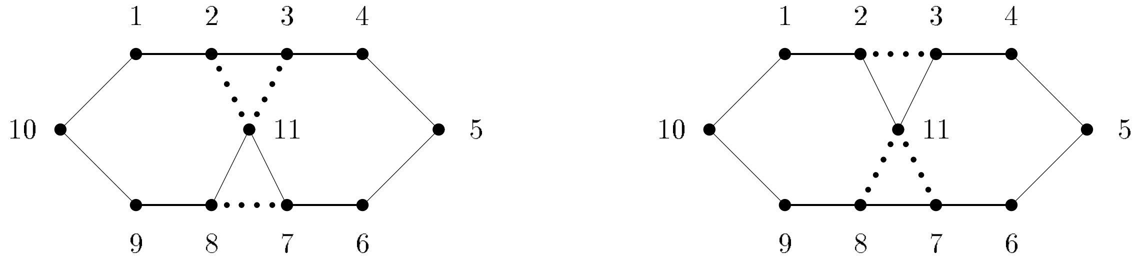
(1) $\{1,3\},\{2,5\}$ and $\{4,6\}$;
(2) $\{1,4\},\{2,5\}$ and $\{3,6\}$;
(3) $\{1,4\},\{2,6\}$ and $\{3,5\}$;
(4) $\{1,5\},\{2,4\}$ and $\{3,6\}$.

The corresponding tours are depicted below.



Case (1) Case (2) Case (3) Case (4)

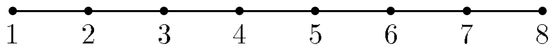
If two of the edges are adjacent, then there is only one way to exchange the edges. This case boils down to the case that one vertex is deleted from the tour and reinserted in another place. If only these types of exchanges are considered, then the algorithm is called vertex-reinsertion, or $2\frac{1}{2}$-opt.



Variable edge-exchange (Lin-Kernighan).
The last exchange algorithm described here is the most sophisticated one. Currently, this algorithm is known as the approximation algorithm that works best for practical TSP problems. It is named after the inventors: Lin and Kernighan. The variable edge-exchange algorithm does not fix the number of edges that are exchanged beforehand, but it determines this number dynamically. We present a simplified version.

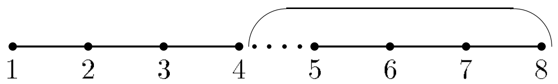
As with all improvement algorithms, an iteration of the Lin-Kernighan algorithm starts with a feasible tour T_0. Suppose that the cities in the tour are numbered in order of their appearance in T_0. Start with deleting an arbitrary edge, say $\{1, n\}$, from T_0; this yields a Hamiltonian path P_0 with endvertices 1 and n. Without loss of generality, vertex 1 is fixed as the head of the path. From among the edges $\{k, n\}$ $(k = 2, \ldots, n-1)$, the one with smallest length is added to the path; suppose this is $\{i, n\}$ (Lin and Kernighan use a somewhat more complicated decision rule). This results in a circuit containing the vertices $\{i, \ldots, n\}$ and a path from 1 to i. The edge $\{i, i+1\}$ is deleted to create a Hamiltonian path P_1; P_1 has 1 and $i+1$ as its endpoints. P_1 can be converted into a tour T_1 by adding the edge $\{1, i+1\}$. The tour T_1 is stored if it has smaller length than T_0. We repeat this process with P_1 instead of P_0 and vertex $i+1$ as the tail of P_1, that is, $i+1$ plays the role that n played in the previous iteration. From among the edges $\{i+1, k\}$ $(k = 2, \ldots, n$, we add the one with smallest length, say edge $\{i+1, j\}$. This results in a circuit and a path again that come together in j. There is only one edge adjacent to j that, after deletion, converts this construction into a *new* Hamiltonian path P_2. P_2 can be converted into a tour T_2 by connecting the endpoints. The tour T_2 is stored if it has smaller length than the tour that is currently best. This process is repeated until an edge that has already been deleted once is selected for addition again.

P_0



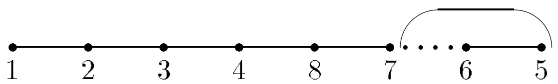
Suppose that the edge $\{4, 8\}$ is selected. To obtain a Hamiltonian path again, we remove the edge $\{4, 5\}$; this yields the Hamiltonian path P_1, which is depicted below. We can turn this path in the Hamiltonian cycle $T_1 \;=\; 1\ 2\ 3\ 4\ 8\ 7\ 6\ 5\ 1$.

P_1



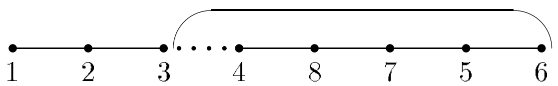
Vertex 5 is the endpoint of the path that is used by the algorithm. Suppose that in this iteration the edge $\{5, 7\}$ is selected; hence, we must remove the ege $\{7, 6\}$. This yields the Hamiltonian path P_2 from which the tour $T_2 \;=\; 1\ 2\ 3\ 4\ 8\ 7\ 5\ 6\ 1$ can be formed.

P_2



We can continue in this fashion by adding the edge $\{6, 3\}$, which implies that we remove $\{3, 4\}$. This yields the Hamiltonian path P_3 from which the tour $T_3 \;=\; 1\ 2\ 3\ 6\ 5\ 7\ 8\ 4\ 1$ can be formed.

P_3



Although there is no worst-case guarantee for this and the previous improvement algorithms other than the guarantee that we have for the initial tour, improvement algorithms, especially the last one, perform much better than constructive algorithms.