Two solvable cases of the Traveling Salesman Problem

Günter Rothe

Dissertation
zur Erlangung des Titels eines Doktors
der technischen Wissenschaften,
eingereicht an der Technisch-
Naturwissenschaftlichen Fakultät
der Technischen Universität Graz

Graz, 1988

Betreuer:
Prof. Dr. Rainer E. Burkard

Adresse:
Technische Universität Graz
Institut für Mathematik
Kopernikusgasse 24
A–8010 Graz

Für Mareike

*Ich danke meiner Frau für die Geduld,
mit der sie mich während der Arbeit
an meiner Dissertation entbehrt hat.*

Abstract

In the Euclidean Traveling Salesman Problem, a set of points in the plane is given, and we look for a shortest closed curve through these lines (a "tour"). We treat two special cases of this problem which are solvable in polynomial time.

The first solvable case is the *N-line Traveling Salesman Problem*, where the points lie on a small number (N) of parallel lines. Such problems arise for example in the fabrication of printed circuit boards, where the distance traveled by a laser which drills holes in certain places of the board should be minimized.

By a dynamic programming algorithm, we can solve the N-line Traveling Salesman Problem for n points in time n^N, for fixed N, i. e., in polynomial time. This extends a result of Cutler (1980) for 3 lines. The parallelity condition can be relaxed to point sets which lie on N "almost parallel" line segments.

The other solvable case concerns the *necklace condition*: A tour is a necklace tour if we can draw disks with the given points as centers such that two disks intersect if and only if the corresponding points are adjacent on the tour. If a necklace tour exists, it is the unique optimal tour.

We give an algorithm which tests in $O(n^{3/2})$ time whether a *given* tour is a necklace tour, improving an algorithm of Edelsbrunner, Rote, and Welzl (1988) which takes $O(n^2)$ time. It is based on solving a system of linear inequalities by the *generalized nested dissection* procedure of Lipton, Rose, and Tarjan (1979). We describe how this method can be implemented with only linear storage.

We give another algorithm which tests in $O(n^2 \log n)$ time and linear space, whether a necklace tour *exists* for a given point set, by transforming the problem to a fractional 2-factor problem on a sparse bipartite graph.

Both algorithms also compute radii for the disks realizing the necklace tour.

Introduction

The *Traveling Salesman Problem* is one of the most important problems in the area of combinatorial optimization. It asks for computing a *tour* in a weighted graph (that is, a cycle that visits every vertex exactly once) such that the sum of the weights of the edges in this tour is minimal. A whole monograph, edited by Lawler, Lenstra, Rinnooy Kan, and Shmoys [1985] has been devoted to this problem.

The traveling salesman problem is known to be NP-hard (see Garey and Johnson [1979] or Johnson and Papadimitriou [1985]) which implies that no algorithm is known currently which finds an optimal tour in polynomial time. There are two approaches to circumvent this difficulty: one is the design of algorithms which compute tours that are close to optimal; the other identifies restricted classes of the problem for which efficient solutions are possible.

In this thesis, we use the second approach, and we identify two such classes with polynomial-time solutions. An overview of many other efficiently solvable cases can be found in Gilmore, Lawler, and Shmoys [1985]. Both classes that we treat are subclasses of the *Euclidean Traveling Salesman Problem* in the plane where the considered graph G is the *complete distance graph* of a finite set P of points in the plane: the vertices of G are the points in P and the weight of an edge between two vertices is the Euclidean distance between the corresponding points. Geometrically, the Euclidean Traveling Salesman Problem can be formulated as the problem of finding a polygon of shortest length whose vertices are the given points.

A *simple* polygon is a polygon in which no two sides have a point in common except for the common endpoint of two adjacent sides. The following lemma is an immediate consequence of the triangle inequality:

Lemma 1. *Unless all points lie on one line the optimal tour is a simple polygon having the given points as vertices.* ■

Although the Euclidean traveling salesman problem is a restricted version of the general traveling salesman problem, it is still NP-hard (see Papadimitriou [1977]). There are a few natural classes of point sets, however, which allow for an efficient construction of optimal tours. For example, the boundary of the convex hull of P forms the unique optimal tour of P if it contains all points of P.

The first solvable case of the Traveling Salesman Problem that we are going to deal with is the *N-line Traveling Salesman Problem*, in which the points lie on N parallel lines, where N is a small number. Such problems arise for example in the fabrication of printed circuit boards, where a laser drills holes in certain places of a board, and the total distance traveled by the laser is to be minimized (cf. Lin and Kernighan [1973]).

By the design, the positions of the holes are naturally aligned on parallel lines.

As will be shown in Chapter 1, the N-line Traveling Salesman Problem can be solved in polynomial time, for fixed N. (The degree of the polynomial is N). The condition that the lines are parallel can be relaxed a little: The points may lie on a set of N "almost parallel" line segments without destroying the properties that are required for our algorithm to work correctly. An exact characterization of those line segments is given in the last section of Chapter 1.

The other class of point sets considered here is based on the notion of the necklace condition:

> Let P be a finite point set in the plane. A tour T of P is a *necklace-tour* if it is the intersection graph of a set S of closed disks centered at the points in P. If a necklace-tour exists, then we say that P *satisfies the necklace condition.*

Figure 1 shows two examples of necklace-tours: a typical "necklace" which illustrates the name of the necklace condition, and a more general example. It has been shown by Sanders [1968] (see Supnick [1970]) that the traveling salesman problem becomes easy if a necklace-tour exists, since a necklace-tour is the unique optimal tour.

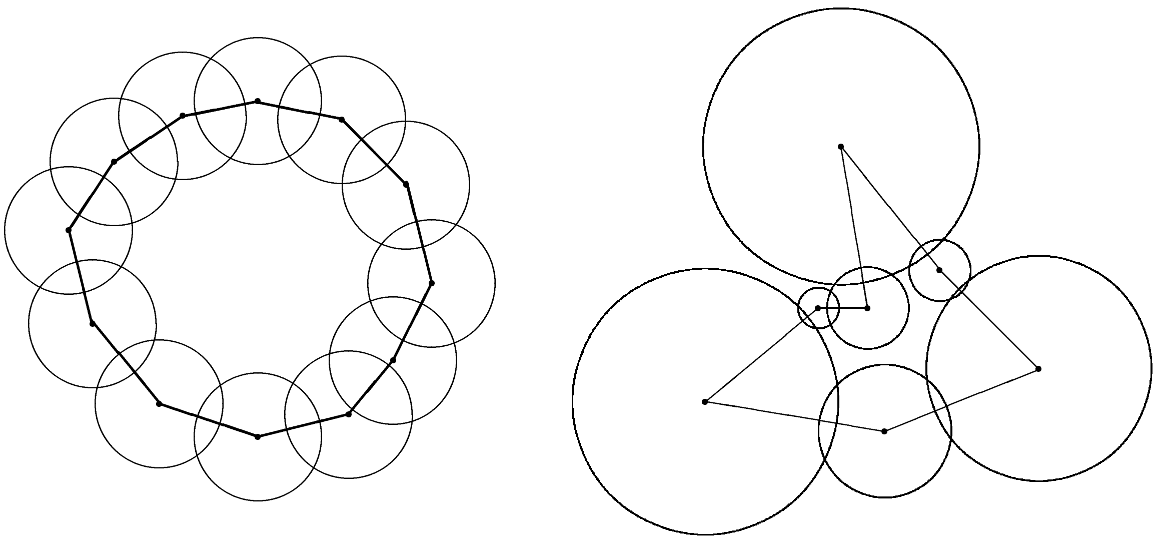

Figure 1. Two necklace tours

The necklace condition is probably not so interesting from the point of view of practical applications, but rather because of the nice characterizations which can be obtained and because of the algorithmic questions that arise naturally: Find out whether a set of points admits a necklace-tour and if yes, find this tour. Chapter 2 deals with this question.

In Section 2.1 we deal with the easier problem of determining whether a *given* tour is a necklace tour. This problem is solved by formulating a system of inequalities whose feasibility has to be checked. This check is carried out by an elimination scheme called *generalized nested dissection.*

In Section 2.2, we consider the problem of testing the necklace condition and finding a necklace tour.

Some results of Chapter 2 (Sections 2.1.1, 2.1.2, 2.1.3.1, and 2.2) originated in cooperation with Herbert Edelsbrunner and Emo Welzl, and they have been published jointly (Edelsbrunner, Rote, and Welzl [1987]).

For an overview of the results obtained in this thesis and their relation to previous results see the individual introductions to the two chapters.

Notational conventions

When we are talking about a subgraph of the complete distance graph, we shall not distinguish between the graph and its edge set. This leads to no confusion since the vertex set is always the set $P = \{p_1, p_2, \ldots, p_n\}$ of given points. In particular, $|G|$ will always denote the number of edges of the graph G, and we will write "$\{p_i, p_j\} \in G$" for "$\{p_i, p_j\}$ is an edge of the graph G".

The Euclidean distance between two points X and Y of the plane will be denoted by $\mathsf{dist}(X, Y)$. For the distance between two given points p_i and p_j we will use the shorter notation $d_{ij} := \mathsf{dist}(p_i, p_j)$.

Chapter 1: The *N*-line Traveling Salesman Problem

1.1. Introduction

In this chapter we will deal with the Traveling Salesman Problem in which all points lie on N parallel (or "almost parallel") lines. We will use a dynamic programming approach to obtain a polynomial algorithm. Our algorithm is an extension of an algorithm by Cutler [1980] for three parallel lines. (For two lines, the problem is trivial.) Cutler also considered the Traveling Salesman *Path* Problem. A similar but easier special case was considered by Ratliff and Rosenthal [1983]: the problem of order-picking in a rectangular warehouse. These authors also used the dynamic programming paradigm for their problem and obtained a linear-time algorithm. Cornuéjols, Fonlupt, and Naddef [1985] extend the result of Ratliff and Rosenthal to the *Steiner Traveling Salesman Problem* for arbitrary undirected series-parallel graphs. In the Steiner Traveling Salesman Problem, we look for a closed path which visits each of a given *subset* of the vertices *at least* once. The algorithm takes linear time.

Another similar algorithm was given in Gilmore, Lawler, and Shmoys [1985], Section 15, for the Traveling Salesman Problem with limited bandwidth. That algorithm also has linear running time (for fixed bandwidth).

Cutler's N-line Traveling Salesman Problem has recently been generalized in a different direction by Deĭneko, van Dal, and Rote [1994]. They considered the problem where the given points lie on the boundary of a convex polygon and on one additional line segment inside this polygon. Clearly, this class of problems contains the 3-line Traveling Salesman Problem as a special case. Moreover, they improved the complexity from $O(n^3)$ to $O(n^2)$.†

Our condition that the lines are parallel can be relaxed, and the algorithm can be applied to points on "almost parallel" lines, which still have the same combinatorial properties with respect to shortest tours.

In the next section we state the exact condition which the lines must fulfill in order that our algorithm is applicable. A characterization of such sets of line segments by forbidden sub-configurations is deferred to Section 1.6, since it is somewhat peripheral to the main course of the paper. In section 2 we continue by stating one simple but essential lemma

† A common generalization of the N-line Traveling Salesman Problem and the convex-hull-and-line Traveling Salesman Problem was considered by Deĭneko and Woeginger [1996]: the convex-hull-and-k-line Traveling Salesman Problem, which has k parallel (or "almost parallel", see the following paragraph) line segments inside the convex hull, whose carrying lines intersect the convex hull in two common edges. The time bound is $O(n^{k+2})$, which corresponds to the time bound in this paper.

that follows from the conditions that we impose on the lines, and we discuss to what extent these conditions are necessary for our algorithm. We also formulate the most elementary facts about the Traveling Salesman Problem, and we define the required notations. In particular, we will define what we mean by a partial solution.

In the third section we derive the main geometric lemma about optimal partial solutions. In Section 1.4, we then formulate a rather standard dynamic programming algorithm, whose correctness is based on that lemma. Finally, in Section 1.5, we analyze the time and space complexity of the algorithm exactly. We can also deal with other metrics than the Euclidean distance. This and other possible extensions and open problems are discussed in the concluding section 1.7.

In this chapter, the terminology will be more of a geometric nature: We will often speak of line segments, polygons, etc. instead of edges, tours, etc.

1.2. Elementary facts, definitions, and notations

We consider the special case where the points lie on N lines, for a small number $N \geq 2$. We shall assume in the rest of this chapter that $N \geq 2$, i. e., not all points lie on a straight line. The algorithm we are going to present works for the case when the lines are parallel, but also more generally when the points lie on N straight line segments fulfilling the following condition:

- No segment is perpendicular to the x-axis.
- For every pair of segments which are not parallel: If we project the two segments and the intersection of their carrying lines onto the x-axis, the projection of the intersection lies always outside the projections of the two segments, never on or between these projections (cf. Figure 2).

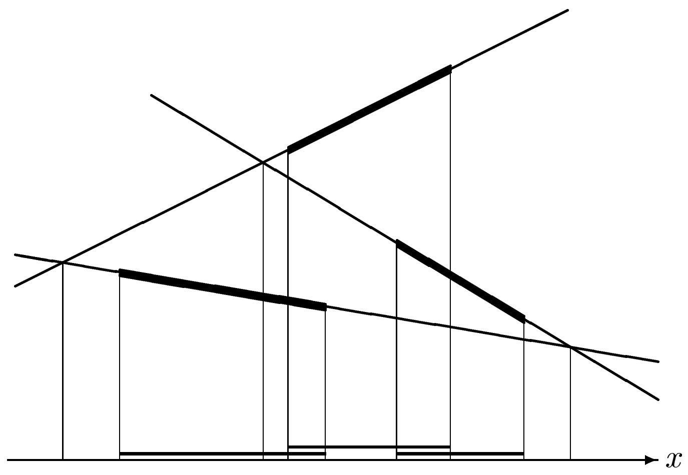

Figure 2.

Three line segments which fulfill
the condition for allowed segments

Of course, the x-axis can be an arbitrary line, but for definiteness, we have chosen this formulation.

This condition allows configurations of segments which are quite non-parallel, such as the one shown in Figure 3a (the half-star) or Figure 3b (the zigzag). On the other hand, the set of segments shown in Figure 4g (the 3-star) or 4e (the triangle) is forbidden. In Section 1.6 we will show that the allowable configurations of segments are precisely those which do not contain a subconfiguration like the ones in Figure 4a–g).

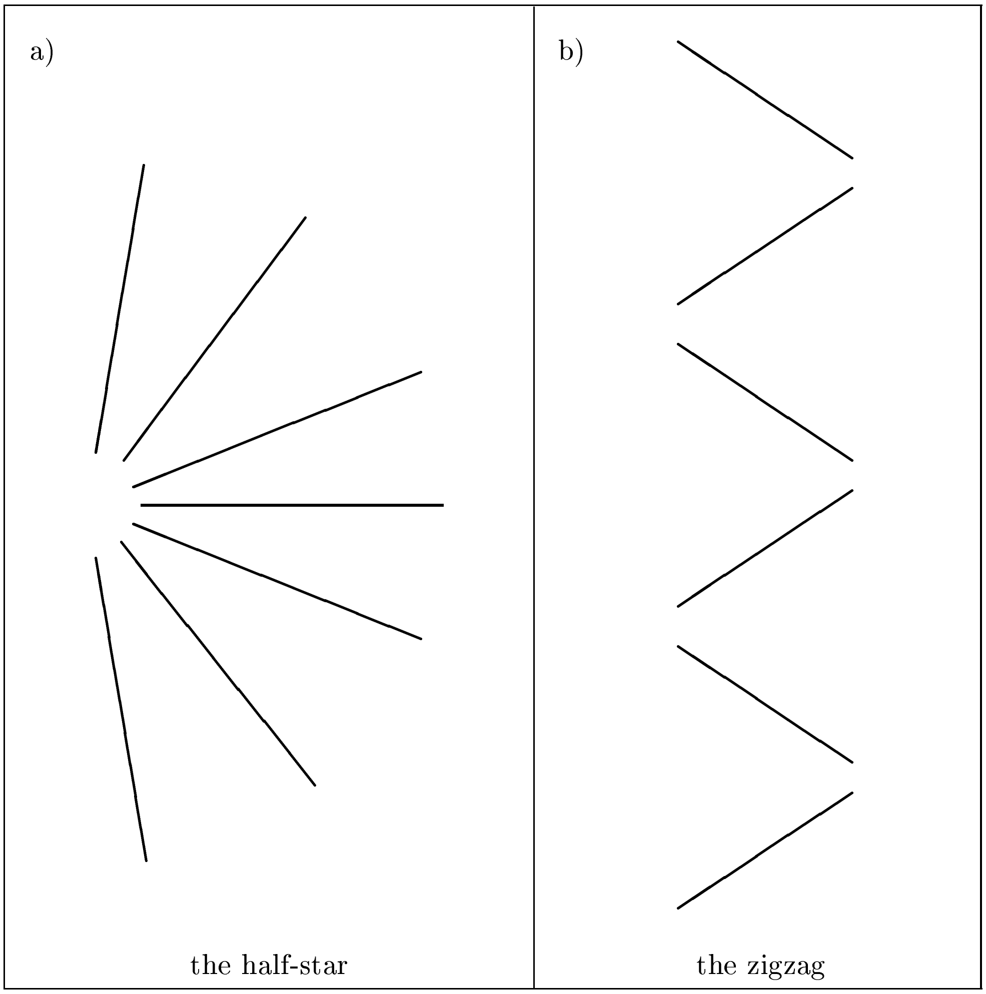

Figure 3. Allowed configurations

This condition has several consequences:

i) The line carrying a segment does not intersect any other segment (cf. Figure 4b). In particular the segments themselves do not intersect (cf. Figure 4a).

ii) It makes sense to speak of "left" and "right" with respect to points that lie on the same segment.

iii) There is a linear ordering of the segments from top to bottom.

Henceforth we shall always assume that the lines are numbered from 1 to N from top to bottom, and that the points on each line are numbered from left to right, from 1 to n_i.

We are goint to exploit one main property that follows from these assumptions, which we formulated in the following lemma. (The notation is already determined by the way how we are going to use the lemma.)

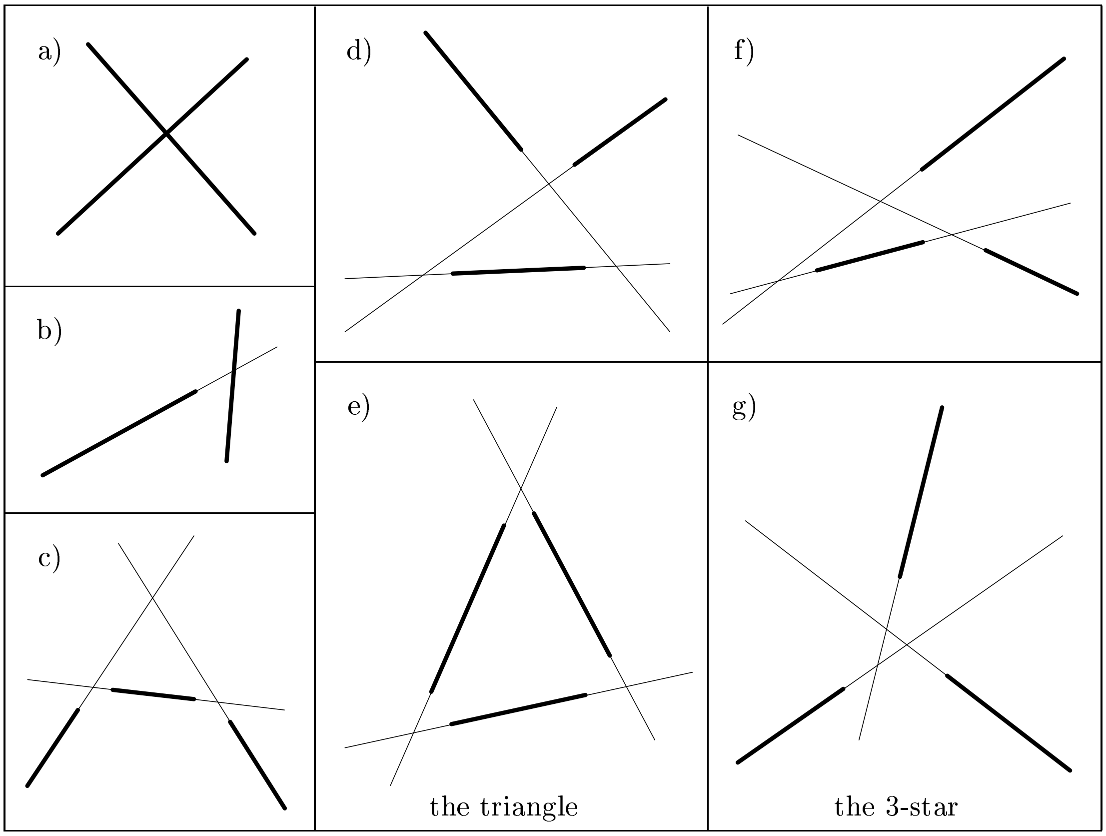

Figure 4. Forbidden configurations

Lemma 2. *Let P be a simple polygon whose vertices are points of the given set. Let l be the bottom-most line that contains a vertex of the polygon, and let k be the top-most line that contains a vertex of the polygon (see Figure 5a.) Assume that P touches each of the lines k and l in more than one point, and let $X_{k'}$ and $B(k)$ denote the left-most and right-most point of the polygon on line k, and similarly, define the points $X_{l'}$ and $B(l)$ on line l. Then the cyclic sequence of these points around the polygon cannot be $X_{k'}$, $B(k)$, $X_{l'}$, $B(l)$ (or the reverse order).*

Proof: Assume that the cyclic order were as given above. Then the simple polygon P together with parts of the lines k and l cound be extended to a planar embedding of the complete bipartite graph $K_{3,3}$, as shown in Figure 5b.[‡] ■

We cannot relax the condition for allowed segment configurations, because then we would not have the ordering of the lines from top to bottom and the consistent ordering of the points from left to right, and it is not obvious how to even formulate an analog of the above properties or of lemma 2.

By a *path* on a set of points we mean an open polygonal line whose vertices are points of the set.

We are going to solve the Traveling Salesman Problem "from left to right", i. e., we will consider partial solutions defined on subsets of points, and we want to extend these

[‡] A different proof of this lemma uses an argument of de Bruijn [1955] about sum of the internal angles of P. (De Bruijn's original argument considered only parallel segments.)

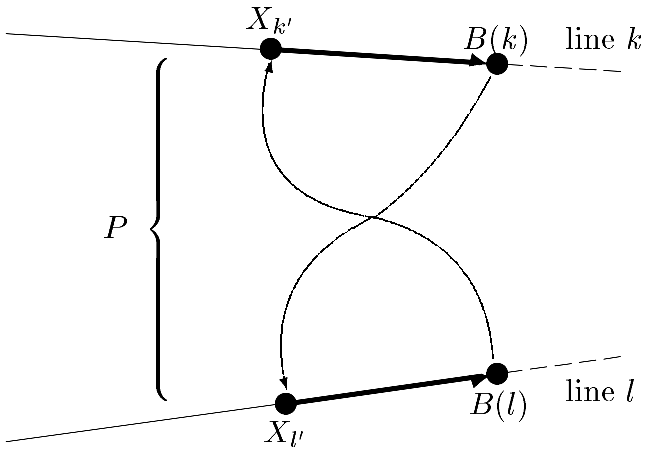

Figure 5a. The polygon P of lemma 2, and the self-intersection in Lemma 1

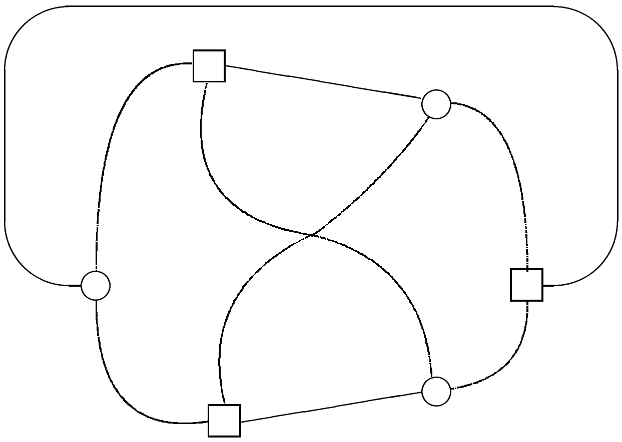

Figure 5b. Extension of the simple polygon P to an embedding of $K_{3,3}$

solutions by adding more and more points on the right. The subsets of points which we consider will now be defined.

Let $(k_1, \ldots, k_N)$ be an N-tuple of integers, $0 \leq k_i \leq n_i$. $P(k_1, \ldots, k_N)$ is the set consisting of the first k_i points of every line. With respect to this sequence, the k_i-th point on line i (it exists only for $k_i > 0$) is called the i-th *boundary point*, denoted by $B(i)$. If $k_i \geq 2$, the $(k_i - 1)$-st point is denoted by $\overline{B}(i)$. Thus, $P(k_1, \ldots, k_N)$ contains the boundary points and the points to their left.

When we look at a tour restricted to a subset $P(k_1, \ldots, k_N)$ of points, we will not get a tour but rather a collection of paths. We want to extend such a partial tour by adding points and edges on the right. Thus, we have to focus our interest on the interface between the partial tour on $P(k_1, \ldots, k_N)$ and the rest of the tour. We shall see that we mainly have to consider the situation near the boundary points, but we have to take care that we don't add edges which connect two endpoints of the same path, forming a subtour which cannot be extended to a complete tour. Thus, in order to know which edges can be added, we have to classify the partial solutions according to the way how the paths connect the boundary points.

Let $S = \{\{i_1, i'_1\}, \{i_2, i'_2\}, \ldots, \{i_m, i'_m\}\}$ be a collection of disjoint two-element subsets of $\{1, \ldots, N\}$.

For non-empty S we define:

$M(S; k_1, \ldots, k_N) =$
the set of all collections $T = \{T_1, \ldots, T_m\}$ of m paths on the set $P(k_1, \ldots, k_N)$, where the j-th path extends between $B(i_j)$ and $B(i'_j)$, for $j = 1, \ldots, m$, and every point in $P(k_1, \ldots, k_N)$ is contained in a path.

In the case where some of the required $B(i_j)$ (or $B(i'_j)$) do not exist because $k_{i_j} = 0$, (or $k_{i'_j} = 0$) $M(S; k_1, \ldots, k_N)$ is empty.

For $S = \emptyset$ we define:

$M(\emptyset; n_1, \ldots, n_N) =$ the set of all tours through the given points.

$M(\emptyset; 0, \ldots, 0) =$ the set consisting only of the "empty tour".

The *length* of T, denoted by $\mathsf{length}(T)$, is the sum of the lengths of its paths.

The elements of the sets $M(S; k_1, \ldots, k_N)$ are called *partial tours*. We call a partial tour *cross-free* if it consists only of paths that do not intersect themselves or each other or if it is a tour which is a simple polygon or if it is the "empty tour". (A turn by 180° in a path of T counts as self-intersection.) Lemma 1 could now be rephrased as saying that an optimal tour is cross-free.

Figure 6a shows a cross-free element T of $M(\{\{2,4\},\{3,5\}\}; 3,5,4,4,4,0)$. Figure 6b shows that a set $M(S; k_1, \ldots, k_N)$ can contain no cross-free element at all. Thus, we do not require the paths in a solution to be disjoint.

The notation $S - \{i_1, j_1\} + \{i_2, j_2\}$ is an abbreviation for $(S \setminus \{\{i_1, j_1\}\}) \cup \{\{i_2, j_2\}\}$.

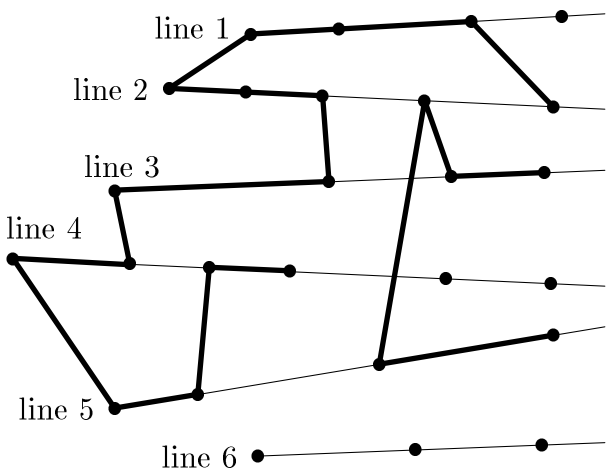

Figure 6a.

A cross-free partial solution of $M(\{\{2,4\},\{3,5\}\}; 3,5,4,4,4,0)$

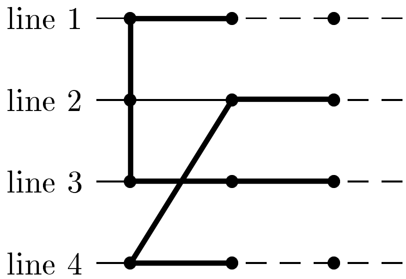

Figure 6b.

$M(\{\{1,3\},\{2,4\}\}; 2,3,3,2)$ contains no partial solutions without crossings.

1.3. Properties of partial solutions

The following lemma states that an optimal partial tour in each set $M(S; k_1, \dots, k_N)$ must contain some edge which is "close to the boundary" of the vertices in $P(k_1, \dots, k_N)$. This will allow us to select the optimal partial tour in each set $M(S; k_1, \dots, k_N)$ from a set of few candidates which consist of a smaller optimal partial tour with an additional edge.

Lemma 2.

A) *If T is a cross-free element of $M(S; k_1, \dots, k_N)$ and $(k_1, \dots, k_N) \neq (0, \dots, 0)$ then T contains one of the following edges:*

(i) *an edge $\{B(i), B(j)\}$, for some i, j with $1 \le i < j \le N$, and $k_i, k_j > 0$, (a "boundary edge")*

or

(ii) *an edge $\{B(i), \overline{B}(i)\}$, for some i with $1 \le i \le N$, where i is contained in a pair of S.*

B) *A cross-free element of $M(\emptyset; n_1, \dots, n_N)$ contains an edge $\{B(i), B(j)\}$, where $1 \le i < j \le N$.*

Example: The set of paths shown in Figure 6a contains a boundary edge between lines 1 and 2, and 3 edges of type (ii) on lines 3, 4, and 5.

Proof: Since part B) is a special case of part A), we have to prove only the first part. We prove it by contradiction, assuming that there is no edge of type (i) or (ii).

If i is contained in a pair of S the edge of T leaving $B(i)$ must go to a different line, otherwise it would be of type (ii). If i is not contained in a pair of S and $k_i > 0$ then there must be two edges in T incident with $B(i)$, but only one of them can go to a point on the same line, otherwise T would not be cross-free. Summarizing, we can say that for every line i with $k_i > 0$ there is an edge from the boundary point $B(i)$ to some point X_i on some other line $f(i) \neq i$ with $k_{f(i)} > 0$. Moreover, X_i is not a boundary point ($X_i \neq B(f(i))$) since we assumed that there is no boundary edge in T.

Now we construct the following polygonal trajectory, which consists alternately of two types of edges (see Figure 7):

- of pieces of the N line segments ("segment edges"), and
- of edges of T ("across edges").

We choose some arbitrary i_0 with $k_{i_0} > 0$, and we start at the point X_{i_0} on line $i_1 := f(i_0)$. Then we go to $B(i_1)$ along the line i_1. Then we take the edge of T to X_{i_1} and set $i_2 := f(i_1)$. Then we go to $B(i_2)$, then to X_{i_2} on line $i_3 := f(i_2)$, and so on.

Since there is only a finite number of points the trajectory must intersect itself. Let Q be the first point where this happens. We claim that the simple polygon P formed by the trajectory between its first and second visit to Q consists alternately of pieces of segments and pieces of edges of T. Since this is true by construction for the whole trajectory, we just have to check the vicinity of Q: If the trajectory leaves Q for the first time along a "segment edge" and enters Q for the second time along an "across edge", or vice versa, the claim is true. Thus we only have to deal with the remaining two cases:

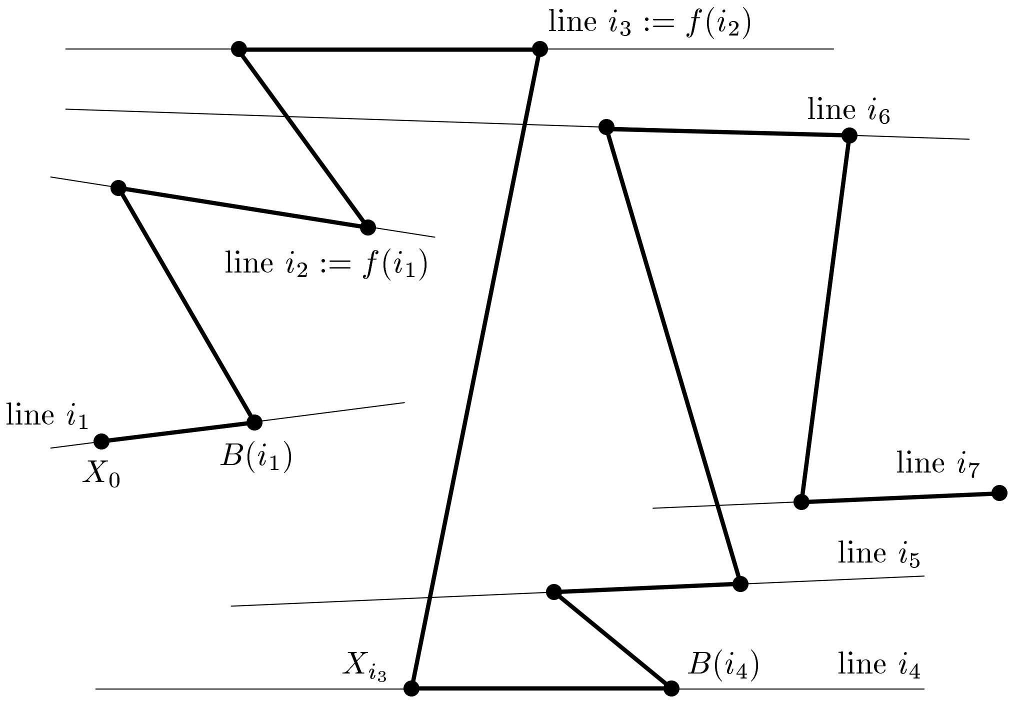

Figure 7. The polygonal trajectory

- The trajectory leaves Q for the first time along an "across edge" $(B(i), X_i)$ of T, and it enters Q for the second time along another "across edge" $(B(j), X_j)$ of T. The only possiblility how this could be consistent with the cross-free property of T is that Q is a common endpoint of the two edges. Q cannot be X_i because the trajectory has to *leave* Q via the edge $(B(i), X_i)$, and similarly, Q cannot be $B(j)$ because the trajectory *enters* Q via the edge $(B(j), X_j)$. The only remaining possibility, $Q = X_j = B(i)$, is excluded because a point X_j is never a boundary point.
- The trajectory leaves Q for the first time along a "segment edge" $\{X_{i_k}, B(i_{k+1})\}$, and it enters Q for the second time along a "segment edge" $\{X_{i_l}, B(i_{l+1})\}$. Two different line segments never cross, and therefore the two edges must lie on the same line $i_{k+1} = i_{l+1}$. In this case, the point Q, which equals X_{i_k}, is redundant as a vertex of P, and the claim is again true.

Now we can apply lemma 2: Let $k = f(k')$ and $l = f(l')$ be the top-most and bottom-most lines that are used by P (Figure 7a). As one traverses P in the direction of the trajectory, one visits $B(k)$ immediately after $X_{k'}$ and after a while $X_{l'}$ and immediately afterwards $B(l)$, and then after a while one returns to $X_{k'}$. But this contradicts lemma 2. (If Q lies on line k or on line l, Q might have to take the place of $X_{k'}$ or $X_{l'}$ in this argument.) ■

When specialized to the case $N = 3$, Lemma 2B is Cutler's "Triangle Theorem", and Lemma 2A corresponds to his "Σ Theorem" (Cutler [1980], Section 4).

Lemma 3.

A) *Let T be a shortest cross-free element of $M(S; k_1, \ldots, k_N)$ $(S \neq \emptyset)$. Then at least one of the following four cases holds (see Figure 8):*

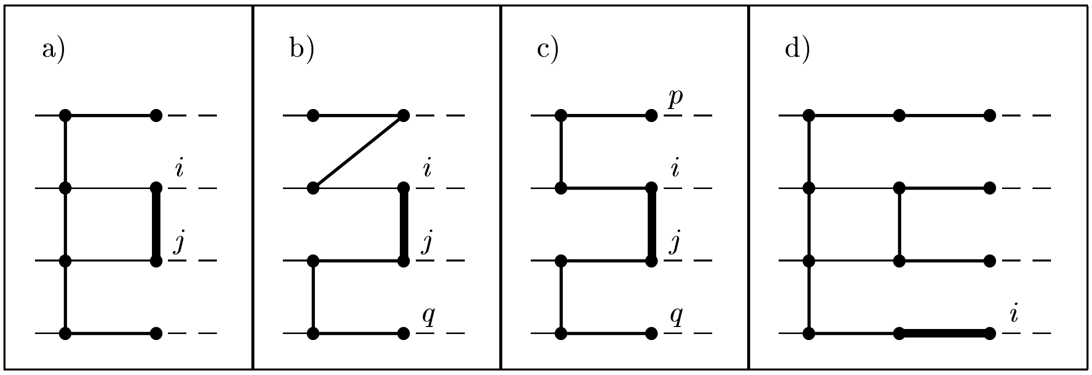

Figure 8. The four possible cases of Lemma 4

(i) *(For some $i \neq j$, T contains an edge $\{B(i), B(j)\}$; $k_i, k_j > 0$:)*

a) $\{i, j\} \in S$:

$$\mathsf{length}(T) = \mathsf{dist}(B(i), B(j)) + \mathsf{length}(T'),$$

where $T' \in M(S - \{i, j\}\,; k_1, \ldots, k_i - 1, \ldots, k_j - 1, \ldots, k_N)$.

b) $\{i, q\} \in S$ *for some* q*, and* j *is not contained in a pair of* S*:*

$$\mathsf{length}(T) = \mathsf{dist}(B(i), B(j)) + \mathsf{length}(T'),$$

where $T' \in M(S - \{i, q\} + \{j, q\}\,; k_1, \ldots, k_i - 1, \ldots, k_N)$.

c) neither i *nor* j *is contained in a pair of* S*:*

$$\mathsf{length}(T) = \mathsf{dist}(B(i), B(j)) + \mathsf{length}(T'),$$

where $T' \in M(S - \{p, q\} + \{i, p\} + \{j, q\}\,; k_1, \ldots, k_N)$ *and* $\{p, q\}$ *is some arbitrary pair of* S.

(ii) *(For some* i*,* T *contains an edge* $\{B(i), \overline{B}(i)\}$*;* $k_i \geq 2$*, and* i *occurs in* S*:)*

d) $\mathsf{length}(T) = \mathsf{dist}(B(i), \overline{B}(i)) + \mathsf{length}(T'),$

where $T' \in M(S; k_1, \ldots, k_i - 1, \ldots, k_N)$.

B) *Let* T *be a shortest simple polygon in* $M(\emptyset; n_1, \ldots, n_N)$*. Then for some* $i < j$*,* T *contains an edge* $\{B(i), B(j)\}$*, and we have:*

$$\mathsf{length}(T) = \mathsf{dist}(B(i), B(j)) + \mathsf{length}(T'),$$

where $T' \in M(\{\{i, j\}\}; n_1, \ldots, n_N)$.

In all cases, T' *is a cross-free element of the respective set.*

Proof: The lemma follows by considering in each case the structure of the partial solution T' which is obtained from T by removing the edge whose existence is implied by the previous lemma. In case (i), the case that i and j are in different pairs of S cannot occur.

■

1.4. The algorithm

We can now set up a dynamic programming recursion corresponding to the equations in Lemma 3 in a straightforward way, involving variables $d(S; k_1, \ldots, k_N)$ corresponding to the sets $M(S; k_1, \ldots, k_N)$.

The starting value of the recursion is $d(\emptyset; 0, \ldots, 0) := 0$.

Let $S = \big\{\{i_1, i'_1\}, \{i_2, i'_2\}, \ldots, \{i_m, i'_m\}\big\}$ and let $l_1, \ldots, l_{N-2m}$ be the indices that do not occur in a pair of S, in some fixed order. Then:

If $S \neq \emptyset$ and $k_{i_j} = 0$ or $k_{i'_j} = 0$ for some $1 \leq j \leq m$, then we set:

$$d(S; k_1, \ldots, k_N) := \infty.$$

Otherwise we set:

$$d(S; k_1, \ldots, k_N) := \min\{\Delta_a, \Delta_{b1}, \Delta_{b2}, \Delta_c, \Delta_{d1}, \Delta_{d2}\},$$

where

$$\Delta_a := \min\big\{ \mathsf{dist}(B(i_j), B(i'_j)) + d(S - \{i_j, i'_j\}; k_1, \ldots, k_{i_j}-1, \ldots, k_{i'_j}-1, \ldots, k_N) \;\big|\; 1 \leq j \leq m \big\}$$

$$\Delta_{b1} := \min\big\{ \mathsf{dist}(B(i'_j), B(l_s)) + d(S - \{i_j, i'_j\} + \{i_j, l_s\}; k_1, \ldots, k_{i'_j}-1, \ldots, k_N) \;\big|\; 1 \leq j \leq m,\ 1 \leq s \leq N-2m \big\}$$

$$\Delta_{b2} := \min\big\{ \mathsf{dist}(B(i_j), B(l_s)) + d(S - \{i_j, i'_j\} + \{i'_j, l_s\}; k_1, \ldots, k_{i_j}-1, \ldots, k_N) \;\big|\; 1 \leq j \leq m,\ 1 \leq s \leq N-2m \big\}$$

$$\Delta_c := \min\big\{ \mathsf{dist}(B(l_s), B(l_t)) + d(S - \{i_j, i'_j\} + \{i_j, l_s\} + \{i'_j, l_t\}; k_1, \ldots, k_N) \;\big|\; 1 \leq s, t \leq N-2m,\ s \neq t,\ 1 \leq j \leq m \big\}$$

$$\Delta_{d1} := \min\big\{ \mathsf{dist}(B(i_j), \overline{B}(i_j)) + d(S; k_1, \ldots, k_{i_j}-1, \ldots, k_N) \;\big|\; 1 \leq j \leq m,\ k_{i_j} \geq 2 \big\}$$

$$\Delta_{d2} := \min\big\{ \mathsf{dist}(B(i'_j), \overline{B}(i'_j)) + d(S; k_1, \ldots, k_{i'_j}-1, \ldots, k_N) \;\big|\; 1 \leq j \leq m,\ k_{i'_j} \geq 2 \big\}$$

(The minimum of an empty set is always taken to be ∞. This ensures that $d(\emptyset; k_1, \ldots, k_N)$ is ∞ unless $k = (0, \ldots, 0)$.)

Finally, as an exception to the above rule:

$$d(\emptyset; n_1, \ldots, n_N) := \min\big\{ \mathsf{dist}(B(i), B(j)) + d(\{\{i, j\}\}; n_1, \ldots, n_N) \;\big|\; 1 \leq i < j \leq N \big\}.$$

We have to establish some order in which these recursions can be computed. We could for example compute the $d(S; k_1, \ldots, k_N)$ in increasing lexicographic order of $(k_1, \ldots, k_N)$. For equal $(k_1, \ldots, k_N)$, the values with larger cardinality of S are computed first. Another possibility would be to compute them in increasing order of $\sum k_i - |S|$, which is equal to the number of edges in the elements of $M(S; k_1, \ldots, k_N)$.

Example: $N = 5$; (a_{ij} denotes the j-th point on line i):

$$
d(\{\{1,3\}\};1,3,5,0,7) = \min \left\{ \begin{array}{ll}
\mathsf{dist}(a_{11},a_{35}) + d(\{\};0,3,4,0,7), & \text{(a)} \\
\mathsf{dist}(a_{11},a_{23}) + d(\{\{2,3\}\};0,3,5,0,7), & \text{(b)} \\
\mathsf{dist}(a_{11},a_{57}) + d(\{\{3,5\}\};0,3,5,0,7), & \text{(b)} \\
\mathsf{dist}(a_{23},a_{35}) + d(\{\{2,3\}\};1,3,4,0,7), & \text{(b)} \\
\mathsf{dist}(a_{35},a_{57}) + d(\{\{3,5\}\};1,3,4,0,7), & \text{(b)} \\
\mathsf{dist}(a_{23},a_{57}) + d(\{\{1,2\},\{3,5\}\};1,3,5,0,7), & \text{(c)} \\
\mathsf{dist}(a_{23},a_{57}) + d(\{\{1,5\},\{2,3\}\};1,3,5,0,7), & \text{(c)} \\
\mathsf{dist}(a_{35},a_{34}) + d(\{\{1,3\}\};1,3,4,0,7) & \text{(d)}
\end{array} \right\}
$$

The first line, corresponding to case (a), can be omitted in this case since $M(\emptyset;0,4,5,0,7)$ is empty and $d(\emptyset;0,4,5,0,7) = \infty$. ■

Lemma 4. *If $M(S;k_1,\ldots,k_N) \neq \emptyset$, then:*

$$
\begin{array}{r} \textit{length of the} \\ \textit{shortest element in} \\ M(S;k_1,\ldots,k_N) \end{array} \;\leq\; d(S;k_1,\ldots,k_N) \;\leq\; \begin{array}{l} \textit{length of the shortest} \\ \textit{cross-free element in} \\ M(S;k_1,\ldots,k_N) \end{array}
$$

Proof: The left inequality is true since $d(S;k_1,\ldots,k_N)$ is always the length of some element from $M(S;k_1,\ldots,k_N)$, and the right inequality follows from Lemma 3 by induction on the recursion. ■

Since for $M(\emptyset;n_1,\ldots,n_N)$ the left and right sides of Lemma 4 are equal (by Lemma 1) we get

Theorem 5. *$d(\emptyset;n_1,\ldots,n_N)$ is the length of the shortest tour.* ■

Remark: Some impossible cases could be excluded from the recursion. For example, if $|S| = 1$ then case (a) need not be considered (like in the previous example) except at the very beginning. However, this would not reduce the running time substantially except for very small N.

1.5. Complexity analysis

Let us now analyze the complexity of the algorithm, both as regards space and time. Let P^N denote the number of different sets S, i. e., the number of sets of disjoint unordered pairs of $\{1, \ldots, N\}$. If we regard the pairs in such a set S as the cycles of a permutation, it is easy to see that P^N equals the number of permutations of N elements whose square is the identity, or equivalently, which are equal to their own inverse.

The numbers P^N can be computed by the recursion:

$$P^{N+1} = P^N + NP^{N-1} \qquad (N \geq 1) \tag{1}$$

from the start values $P^0 = P^1 = 1$. This recursion was given in Rothe [1800], p. 282. It can be proved by splitting the P^{N+1} sets S into those where the element $N+1$ is not contained in a pair and into those where it forms a pair with one of the N other elements. (The very same recursion, but with different start values, occurs in Gilmore, Lawler, and Shmoys [1985], p. 138, where it describes the number of equivalence classes of partial tours for the bandwidth-constrained Traveling Salesman Problem.)

The following table shows the first few values of P^N: (The meaning of the last column will be explained later.)

N	P^N	$\mathsf{TIME}'(N)$
1	1	0
2	2	3
3	4	15
4	10	72
5	26	300
6	76	1,290
7	232	5,418
8	764	23,520
9	2,620	102,672
10	9,496	461,700
11	35,696	2,107,380
12	140,152	9,876,768
13	568,504	47,127,600

Chowla, Herstein, and Moore [1951] proved the following asymptotic expression for P^N:

$$P^N \sim \left(\frac{N}{e}\right)^{N/2} e^{\sqrt{N}} \frac{1}{\sqrt[4]{4e}}.$$

Very roughly, P^N is $\sqrt{N!}$. A more exact approximation with additional terms of higher order was proved by Moser and Wyman [1955] (cf. also Knuth [1973], pp. 65–67).

The storage requirement for the algorithm is now

$$P^N \cdot \prod_{i=1}^{N} (n_i + 1)\,.$$

Thus, for fixed N, the storage requirement is

$$O\left(\prod_{i=1}^{N}(n_i+1)\right) = O\left(\prod_{i=1}^{N} n_i\right) = O(n^N),$$

if we denote the total number of points by $n = \sum_{i=1}^{N} n_i$.

If n is fixed then the maximum of the left side in the above equation is achieved for $n_i = n/N$; thus, using the inequality $n_i + 1 \le 2n_i$, the constant of the O-notation in the expression $O(n^N)$ is at most

$$\left(\frac{2}{N}\right)^N \cdot P^N = \left(\frac{2}{N}\right)^N \left(\frac{N}{e}\right)^{N/2} e^{\sqrt{N}} \frac{1}{\sqrt[4]{4e}} = O\left(\frac{1}{(N/4)^{N/2}} \cdot \frac{1}{e^{N/2-\sqrt{N}}}\right),$$

which decreases quite fast as N increases.

For analyzing the time complexity, let us now establish the complexity of one step of the recursion: If S contains m pairs, then there are at most m, $2m(N-2m)$, $(N-2m)(N-2m-1)m$, and $2m$ terms corresponding to cases (a), (b), (c), and (d), respectively. Therefore, the total number of terms which are necessary for computing $d(S; k_1, \ldots, k_N)$ is the sum of these four expressions, which is $m(3+N+N^2) + m^2(-4N-2) + 4m^3$.

Let P_m^N denote the number of sets S containing exactly m disjoint unordered pairs of $\{1, \ldots, N\}$. It is equal to the number of permutations with m cycles of length 2 and $N-2m$ cycles of length 1. Therefore, we have

$$P_m^N = \frac{N!}{2^m m!\,(N-2m)!}, \qquad \text{for } 0 \le m \le N/2. \tag{2}$$

Neglecting the boundary cases, the time complexity is thus

$$\mathsf{TIME}(N) = \mathsf{TIME}'(N) \cdot \left(\prod_{i=1}^{N}(n_i+1)\right),$$

where $\mathsf{TIME}'(N)$, the time for evaluating $d(S; k_1, \ldots, k_N)$ for all sets S for some fixed N-tuple $(k_1, \ldots, k_N)$, is given as follows:

$$\mathsf{TIME}'(N) = \sum_m P_m^N \cdot \left(m(3+N+N^2) + m^2(-4N-2) + 4m^3 \right). \tag{3}$$

(By observing that $P_m^N = 0$ for $m < 0$ or $m > N/2$, we may simplify matters by letting the summation index m vary over all integers.)

The terms involving the variable m in this sum can be eliminated by using the following formulas, which follow directly from (2):

$$\begin{aligned}
P_m^N\, m &= \frac{N(N-1)}{2}\, P_{m-1}^{N-2} \\
P_m^N\, m(m-1) &= \frac{N(N-1)(N-2)(N-3)}{4}\, P_{m-2}^{N-4} \\
P_m^N\, m(m-1)(m-2) &= \frac{N(N-1)(N-2)(N-3)(N-4)(N-5)}{8}\, P_{m-3}^{N-6}.
\end{aligned}$$

After this we can carry out the summation over m, using the identity $P^N = \sum_m P^N_m$, and express $\mathsf{TIME}'(N)$ in terms of N, P^{N-2}, P^{N-3}, ..., and P^{N-6}. Repeated application of the recursion (1) leads then to the following short expression:

$$\mathsf{TIME}'(N) = \frac{N(N-1)}{2}\left(P^N + P^{N-2}\right).$$

(The calculation is carried out in detail in the original version of the technical report Rote [1988].) The values of $\mathsf{TIME}'(N)$ are tabulated in the above table.

Thus, for fixed N, the time complexity is again

$$O\left(\prod_{i=1}^{N}(n_i+1)\right) = O\left(\prod_{i=1}^{N} n_i\right) = O\big(n^N\big).$$

Arguing as in the case of the storage requirement, we find that the constant of the O-notation in the expression $O(n^N)$ is at most

$$\left(\frac{2}{N}\right)^N \cdot \binom{N}{2}\left(P^N + P^{N+2}\right) = O\left(\left(\frac{2}{N}\right)^N N^2 \left(\frac{N}{e}\right)^{N/2} e^{\sqrt{N}}\right)$$
$$= O\left(\frac{1}{(N/4)^{N/2-2}} \cdot \frac{1}{e^{N/2-\sqrt{N}}}\right).$$

We summarize our results in the following

Theorem 6. *The N-line Traveling Salesman Problem with n_1, n_2, ..., n_N points on the lines 1, 2, ..., N can be solved in space*

$$O\left(P^N \cdot \prod_{i=1}^{N}(n_i+1)\right)$$

and time

$$O\left(P^N \cdot N^2 \prod_{i=1}^{N}(n_i+1)\right),$$

where the numbers P^N are defined by the recursion (1). *For fixed N, and for a total number of n points, the space and time complexities are thus*

$$O\big(n^N\big).$$

■

1.6. Characterization of "quasi-parallel" line segments

In Section 1.2, the following property was required of a set of line segments in order that our algorithm could be applied:

> No segment is perpendicular to the x-axis, and for any two segments which are not parallel the projection of the intersection of the two lines carrying the segments onto the x-axis lies always outside the projections of the two segments.

We call a set of line segments *quasi-parallel*, if they can be rotated in such a way that this property is fulfilled, in other words, if an appropriate x-axis can be found. In this section, we give a characterization of this property.

First we have to introduce some terminology:

By an *orientation* of a line we mean an assignment of the label "left" to one direction of the line and the label "right" to the other direction. An *oriented line* is a line together with an orientation. When we draw an oriented line the orientation will be indicated by an arrow pointing in the "right" direction.

By an *oriented segment* we mean a segment a with an orientation of the line $g(a)$ carrying the segment.

When we project an oriented line on another line which is not perpendicular to it we get a *corresponding orientation* of the second line in a natural way.

Now we call two oriented segments a and b *oriented consistently* if the intersection of the carrying lines $g(a)$ and $g(b)$ either lies both on the left side of a on $g(a)$ and on the left side of b on $g(b)$ or on the right side of a resp. b on both $g(a)$ and $g(b)$. (In case a and b are parallel, the orientations have to be the same in order to be consistent.)

We can now rephrase the above definition of quasi-parallelness in the terminology just introduced as follows:

> There is an oriented line l (this line corresponds to the x-axis in the previous formulation), not perpendicular to any segment, such that for the corresponding orientations of the segments obtained by projection from the orientation of l, any pair of non-parallel segments a and b is oriented consistently.

Theorem 3. *Let a set of at least 3 segments be given, and let g_0 be any fixed segment of this set. Then the set of segments is quasi-parallel if and only if every subset of three segments containing g_0 is quasi-parallel.*

Proof: First of all, it is clear that consequence (i) of Section 1.2 holds, i. e., the line carrying a segment does not intersect any other segment, and in particular, the segments themselves do not intersect.

Now fix any orientation of g_0. Then there is only one possible orientation of every other segment which is consistent with the orientation of g_0.

We have to show two things:

(a) There is an oriented line l such that the orientations thus constructed correspond by projection to the orientation of l.

(b) Every pair of non-parallel segments a and b (different from g_0) is also oriented consistently.

For proving (b), let's assume that two segments a and b are not oriented consistently. Then the orientation of b resulting from the orientation of g_0 is different from the orientation of b resulting from the orientation of a resulting from the orientation of g_0; thus, even for the 3-set $\{g_0, a, b\}$, it would be impossible to orient the segments consistently.

Now we still have to prove (a). For each segment the possible oriented directions of l form an open half-circle (cf. Figure 9). We know that the half-circle belonging to g_0 has a non-empty intersection with every two other half-circles. We intersect the other half-circles with the half-circle belonging to g_0 and consider only the results, as parts of the half-circle belonging to g_0, or, equivalently, as intervals. We know that any two of these intervals have a non-empty intersection. From this it follows, by Helly's Theorem, that the intersection of all intervals is non-empty. Therefore there is a possible orientation of l yielding the constructed orientations of the segments. ■

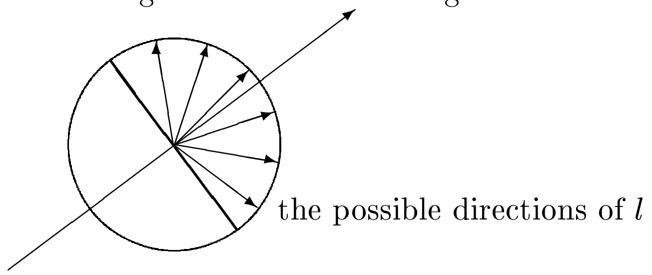

Figure 9.
The possible directions of l for
a given direction of a segment

An $O(N^2)$-algorithm for testing whether N line segments are quasi-parallel follows in a straightforward manner. It is possible to reduce this time to $O(N \log N)$, but since this is by far surpassed by the complexity of solving the problem, this is not so interesting.

By a simple case analysis, one can determine all configurations of three or fewer line segments which are not quasi-parallel, and thus one obtains the following theorem as an easy corollary:

Theorem 8. *A set of segments is quasi-parallel if and only if it does not contain a subset of segments which looks like one of the seven* types of configurations in Figure 4.* ■

---

* In the original version I had only six types of configurations. I thank Gerhard Woeginger for pointing out that the configuration in Figure 4c was missing.

1.7. Conclusion

From a higher standpoint our approach to the N-line Traveling Salesman Problem can be viewed in the following way: A partial tour was defined to be a certain subset of the edges of a tour. We have grouped the partial tours into equivalence classes with respect to the edge sets which can be added to complete the tour: Two partial tours T_1 and T_2 are equivalent if, for all sets T', $T_1 \cup T'$ is a tour if and only if $T_2 \cup T'$ is a tour. This allowed us to forget all partial solutions except the best one in each equivalence class. With little effort, we have now been able to compute the best partial solution in each equivalence class in a systematic way.

Exactly the same paradigm has been followed by Ratliff and Rosenthal [1983] in their solution of the order-picking problem, and by Gilmore, Lawler, and Shmoys [1985] for the bandwidth-limited problem.

It is in principle not difficult to extend this approach to the Taveling Salesman *Path* problem, which requires to find a shortest, not necessarily closed curve containing the given set of points. It is necessary to modify the notion of a partial solution and to establish a corresponding analog of Lemma 3.

Our algorithm also applies to Traveling Salesman Problems in the plane with other metrics than the Euclidean distance, for example the L_1 metric (Manhattan metric) and the L_∞ metric (maximum metric). These metrics are important for the class of manufacturing problems that were mentioned in the introduction. In fact, the only property of the Euclidean distance that we have used is the triangle inequality, which was necessary for the cross-free property of Lemma 1. It is clear that there is always an optimal tour such that the corresponding polygon is a simple polygon, for any symmetric distance function that fulfills the triangle inequality. Thus, the only thing in the algorithm that has to be changed is the computation of $\mathsf{dist}(X, Y)$.

A question which we have not pursued so far is the following: Can our results be applied to construct heuristic algorithms in cases when there is no set of few parallel straight lines containing the points? One might only require that the points lie in the vicinity of the lines; or one might reduce the number of partial solutions by excluding "unlikely" sets $P(S; k_1, \ldots, k_N)$, whose boundary points are distributed in a wide range between the left-most and the right-most extremes, thus speeding up the algorithm.

Chapter 2: The necklace condition

Introduction

Although the necklace condition was introduced as early as 1968 (cf. Supnick [1968]), the algorithmic questions associated with it have not attracted the attention of researchers. It was not even noticed in the survey paper of Gilmore, Lawler, and Shmoys [1985] on well-solved special cases of the Traveling Salesman Problem.

These questions were raised for the first time in the paper of Edelsbrunner, Rote, and Welzl [1987], from which some parts of this chapter are taken. Their algorithms can solve the problem of testing whether a given tour is a necklace tour in $O(n^2)$ time, and they can test whether a point set has a necklace tour in $O(n^2 \log n)$ time. In both cases, if a solution exists, the tour and radii which realize it can be found in the same time complexity. All algorithms require only $O(n)$ space.

In the Section 2.1 we will deal with the first question mentioned above: Given a tour, test whether it is a necklace tour. Initially we follow the approach of Edelsbrunner, Rote, and Welzl [1987]: The necklace condition specifies exactly which pairs of disks must intersect and which pairs of disks must not intersect. This condition easily translates into a system of inequalities for the radii of the disks. There are $\Theta(n^2)$ inequalities, one for each pair of points.

In Section 2.1.1 we show that this system of inequalities can be reduced to an equivalent system with a smaller number of inequalities: Firstly, the restriction that the radius of a disk be a positive number can be omitted. (This constraint comes from the geometric concept of a disk; it is not necessary for the optimality of necklace tours, anyway.) Secondly, not all pairs of points have to be considered: it is sufficient to consider those pairs which corresponds to edges in a certain graph $G^{(2)}$ which is defined from geometric properties of the point set.

In Section 2.1.2 we show that this graph has only $O(n)$ edges, and hence the number of inequalities of the system that has to be solved is $O(n)$. Furthermore, we describe how the graph $G^{(2)}$ can be constructed in $O(n \log n)$ time.

In section 2.1.3.1, the system is transformed in such a way that it becomes equivalent to a single-source shortest path problem. In Edelsbrunner, Rote, and Welzl [1987], this shortest path problem was solved by the Bellman-Ford algorithm, leading to a time complexity of $O(n^2)$. However, by using *generalized nested dissection* as introduced by Lipton, Rose, and Tarjan [1979], we can reduce the complexity to $O(n^{3/2})$.

We present an outline of both procedures in Section 2.1.3.2, as well as an overview of other related solution methods for systems of linear inequalities.

In order to apply generalized nested dissection we need a further property of the graphs $G^{(2)}$, namely that they can be separated into two parts of approximately the same size by removing a set of $O(\sqrt{n})$ vertices (a "separator"). This property is proved in Section 2.1.3.3, using a corresponding property for planar graphs.

In Section 2.1.3.4, we describe the generalized nested dissection method for the case of $\sqrt{n}$-separators in detail. The generalized nested dissection method as described in Lipton, Rose, and Tarjan [1979] takes $O(n^{3/2})$ time and $O(n \log n)$ space in the case of $\sqrt{n}$-separators. They mention the possibility that the space requirement can in principle be reduced to $O(n)$ only parenthetically. Therefore we give a detailed account of the techniques that are required for reducing the storage to $O(n)$ while maintaining the time complexity of $O(n^{3/2})$. For the analysis of the time complexity we can essentially rely on the results of Lipton, Rose, and Tarjan [1979], whereas the space complexity is derived anew.

Section 2.2 presents the solution of Edelsbrunner, Rote, and Welzl [1987] for the problem of finding a necklace tour if it exists. Graphs for which a necklace tour exists can be characterized by certain properties of the optimal solution of a corresponding *fractional 2-factor problem*, which is described in Section 2.2.1. Section 2.2.2 shows that the fractional 2-factor problem can be transformed into a transportation problem. In $O(n^2 \log n)$ time, one can find an optimal solution of this problem. If a necklace tour exists, this solution corresponds directly to the necklace tour.

A short summary of the results of this chapter is given in Section 2.3.

2.1. Testing whether a given tour is a necklace tour

A tour F of G is a necklace tour if there are radii r_i fulfilling the following condition:

$$\text{(R)} \quad \begin{cases} r_i + r_j \geq d_{ij}, & \text{if } \{p_i, p_j\} \in F \qquad \text{(R.1)} \\ r_i + r_j < d_{ij}, & \text{if } \{p_i, p_j\} \notin F \qquad \text{(R.2)} \\ r_i > 0, & \text{for } 1 \leq i \leq n, \qquad \text{(R.3)} \end{cases}$$

The numbers r_i are said to *realize* F.

A tour T in a graph is a spanning subgraph with the following two properties:

(i) Every vertex has degree 2.

(ii) T is connected.

A spanning subgraph fulfilling the first property is called a *2-factor* (or a *2-matching*) of the graph. Virtually all results about necklace tours can easily be extended to 2-factors. Sometimes they are even easier to formulate for 2-factors than for tours. Hence we define a *necklace 2-factor* as a 2-factor F which fulfills the conditions (R).

In Section 2.1.1, we shall prove that a small subset of the inequalities (R) is sufficient for the necklace property. To define this subset, we have to introduce some notations:

Let m be a positive integer smaller than n. For each point $p \in P$, we let $d^{(m)}(p)$ denote the distance from p to the m-nearest neighbor of point p, that is, $d^{(1)}(p)$ is the distance from p to its nearest neighbor, $d^{(2)}(p)$ is the distance to the 2^{nd}-nearest neighbor, and

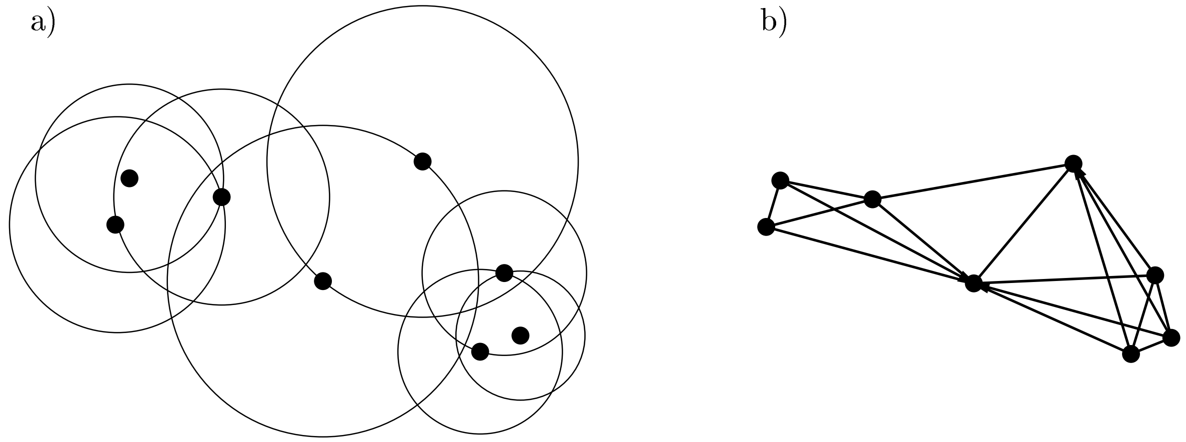

Figure 10. A set of disks $D^{(2)}(p)$ and the graph $G^{(2)}$

so on. We define $D^{(m)}(p)$ to be the closed disk centered at p with radius $d^{(m)}(p)$. The graph $G^{(m)}$ is the intersection graph of these disks: There is an edge between two points if and only if the two disks intersect, i. e., if $d^{(m)}(p_i) + d^{(m)}(p_j) \geq d_{ij}$. (cf. Figure 10).

In $G^{(m)}$, every vertex has at least degree m; therefore, the number of edges is at least $mn/2$.

Theorem 9 in Section 2.1.1 says that the relations (R) have only to be checked for edges in $G^{(2)}$. In Section 2.1.2, we shall prove that $G^{(2)}$ is a sparse graph, and we shall describe how $G^{(2)}$ can be constructed. Section 2.1.3 discusses the way how the reduced system can be solved. The results are summarized in Section 2.3.

2.1.1. Systems of linear inequalities characterizing necklace tours

In this section we are going to prove that we can ignore the positivity constraints (R.3) and that we only need to check edges of $G^{(2)}$, i. e., we can replace (R.2) by

$$r_i + r_j < d_{ij}, \quad \text{if } \{p_i, p_j\} \in G^{(2)} \setminus F. \qquad \text{(R.2}'\text{)}$$

The proof is fairly technical and involved.

Theorem 9. *Let F be a 2-factor. Then the following statements are equivalent:*

(i) *The system* (R.1), (R.2), (R.3) *is solvable (i. e., F is a necklace 2-factor).*

(ii) *The system* (R.1), (R.2′) *is solvable.*

Proof: The non-trivial part is the conclusion from (ii) to (i). We proceed in several steps.

Lemma 10. *Let r_i be numbers fulfilling* (R.1), (R.2′) *for a 2-factor F. If $r_i \leq 0$, $d_{ij} \leq r_j$, and $d_{ij} \leq d^{(2)}(p_j)$, then $\{p_i, p_j\} \in F$.*

Proof: (by contradiction.) Assume that there is a point p_i with non-positive r_i, $d_{ij} \leq r_j$, $d_{ij} \leq d^{(2)}(p_j)$ and $\{p_i, p_j\} \notin F$, for some $j \neq i$. and let p_i be the point with smallest value r_i among these, and let p_k and $p_{k'}$ be the neighbors of p_i in F (see Figure 11). We have

$$r_i + r_k \geq d_{ik} \qquad \text{and} \qquad r_i + r_{k'} \geq d_{ik'}, \tag{4}$$

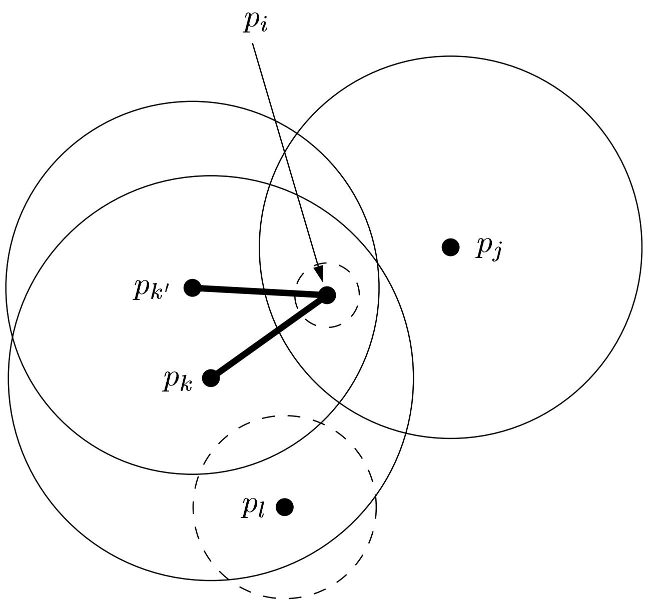

Figure 11. Proof of Lemma 10.

p_i and p_j constitute a counterexample for Lemma 10. Disks with negative radius are shown with broken lines.

Abridged version of the proof: If $d^{(2)}(p_k) \geq d_{ik}$ and $d^{(2)}(p_{k'}) \geq d_{ik'}$, then the disks $D^{(2)}(p_k)$, $D^{(2)}(p_{k'})$, and $D^{(2)}(p_j)$, as well as the disks around p_k, $p_{k'}$, and p_j with radii r_k, $r_{k'}$, and r_j, resp., would contain the common point p_i, which would yield too many edges incident to p_k in F.

Now let $d_{ik} \geq d^{(2)}(p_k)$. $D^{(2)}(p_k)$ contains 2 points, one of which — p_l — is not connected to p_k. We have $d_{ki} + (-r_i) \leq r_k$ but $d_{kl} + (-r_l) < r_k$ (see figure). Together with $d_{kl} \leq d_{ki}$ this implies $-r_i < -r_l$.

and hence

$$r_k \geq d_{ik} \qquad \text{and} \qquad r_{k'} \geq d_{ik'}.$$

Next, we show that $d_{ik} > d^{(2)}(p_k)$ or $d_{ik'} > d^{(2)}(p_{k'})$: Otherwise we would get, using the triangle inequality:

$$\begin{aligned} d^{(2)}(p_k) + d^{(2)}(p_{k'}) &\geq d_{ik} + d_{ik'} \geq d_{kk'}, && \text{and hence } \{p_k, p_{k'}\} \in G^{(2)}, \text{ and} \\ d^{(2)}(p_k) + d^{(2)}(p_j) &\geq d_{ik} + d_{ij} \geq d_{kj}, && \text{and hence } \{p_k, p_j\} \in G^{(2)}. \end{aligned}$$

Furthermore, we would have

$$\begin{aligned} r_k &\geq r_k + r_i \geq d_{ik}, \text{ and} \\ r_{k'} &\geq r_{k'} + r_i \geq d_{ik'}, \end{aligned}$$

by (R.1), and hence

$$\begin{aligned} r_k + r_{k'} &\geq d_{ik} + d_{ik'} \geq d_{kk'}, \text{ and} \\ r_k + r_j &\geq d_{ik} + d_{ij} \geq d_{kj}, \end{aligned}$$

by the triangle inequality. Since $\{p_k, p_{k'}\} \in G^{(2)}$ and $\{p_k, p_j\} \in G^{(2)}$, as was just shown above, this would imply that $\{p_k, p_{k'}\} \in F$ and $\{p_k, p_j\} \in F$, by (R.2′). But since also $\{p_k, p_i\} \in F$, this would contradict the 2-factor property of F.

Thus, we proved that at least one neighbor of p_i (say, p_k) fulfills

$$d_{ik} > d^{(2)}(p_k). \tag{5}$$

There are at least two points p_l with $d_{kl} \leq d^{(2)}(p_k)$, by the definition of $d^{(2)}(p_k)$. Not both of them can be neighbors of k in F, since for one of the two neighbors of k, namely i, we have $d_{ki} > d^{(2)}(p_k)$. Hence there is a point p_l with

$$d_{kl} \leq d^{(2)}(p_k) \qquad \text{and} \qquad \{p_k, p_l\} \notin F. \tag{6}$$

From that inequality follows $\{p_k, p_l\} \in G^{(2)}$ and hence, by (R.2′),

$$r_k + r_l < d_{kl}.$$

From (4), (5), and (6), we get

$$r_l < r_i \leq 0,$$

and moreover we have also

$$d_{lk} \leq r_k + r_l \leq r_k,\ d_{lk} \leq d^{(2)}(p_k), \text{ and } \{p_k, p_l\} \notin F.$$

Thus, the point p_l, together with p_k, constitutes another counterexample to our lemma, contradicting the choice of p_i as the counterexample with smallest r_i at the beginning of the proof. ■

Lemma 11. *If a set of values r_i, $i = 1, 2, \ldots, n$, fulfills* (R.1) *and* (R.2′) *then the values*

$$r_i' := \begin{cases} d^{(2)}(p_i), & \text{if } r_i > d^{(2)}(p_i)\ , \\ 0, & \text{if } r_i < 0\ , \quad \text{and} \\ r_i, & \text{otherwise}\ , \end{cases}$$

also fulfill (R.1) *and* (R.2′).

Proof: We have now $0 \leq r_i' \leq d^{(2)}(p_i)$, for all i. In the proof below, we distinguish two cases, namely the case that an edge belongs to F and the case that it belongs to $G^{(2)} \setminus F$. We treat the second case first.

(i) If $\{p_i, p_j\} \in G^{(2)} \setminus F$, then we have $r_i + r_j < d_{ij}$. The corresponding inequality

$$r_i' + r_j' < d_{ij}$$

can be violated only if one previous variable (say, r_i) is negative, which implies $r_i' = 0$. In this case, $r_j \geq d_{ij}$ would imply $r_j \geq r_j' \geq d_{ij}$ and $d^{(2)}(p_j) \geq r_j' \geq d_{ij}$, which would contradict Lemma 10. Thus, we have

$$r_j' < d_{ij} \text{ and } r_i' + r_j' = r_j' < d_{ij}.$$

(ii) If $\{p_i, p_j\} \in F$, then we have $r_i + r_j \geq d_{ij}$. The corresponding inequality

$$r_i' + r_j' \geq d_{ij}$$

can become violated only if one variable r_i or r_j (say, r_i) is greater than $d^{(2)}(p_i)$, which implies $r'_i = d^{(2)}(p_i)$. We have $d^{(2)}(p_i) \geq d_{ik}$ for at least 2 points p_k. Hence,

$$r'_i + r'_k \geq r'_i = d^{(2)}(p_i) \geq d_{ik} \text{ and}$$

$$d^{(2)}(p_i) + d^{(2)}(p_k) \geq d^{(2)}(p_i) \geq d_{ik},$$

which implies $\{p_i, p_k\} \in G^{(2)}$, for at least 2 points p_k. We have $\{p_i, p_k\} \in F$, for each of these points, since, otherwise, $r'_i + r'_k < d_{ik}$, by part (i) of this proof. But there are only 2 points p_k with $\{p_i, p_k\} \in F$. Thus,

$$r'_i + r'_k \geq d_{ik}$$

holds for all points p_k with $\{p_i, p_k\} \in F$ and therefore also for point p_j. ■

Now, we come to the conclusion of the proof of Theorem 9. By the preceding lemma, we have transformed a solution of (R.1), (R.2′) into a solution fulfilling the additional constraints

$$0 \leq r_i \leq d^{(2)}(p_i).$$

Now, for $\{p_i, p_j\} \notin G^{(2)}$, we get the remaining inequalities of (R.2)

$$r_i + r_j \leq d^{(2)}(p_i) + d^{(2)}(p_j) < d_{ij}$$

for free. Let $\delta = min\,\{\, d_{ij} - r_i - r_j \mid \{p_i, p_j\} \notin F\,\} > 0$. Then $r''_i := r_i + \delta/3$ still fulfills all inequalities (R.1) and (R.2) and, in addition, we have $r''_i > 0$. ■

The following theorem is important because in the next section we will show that $G^{(2)}$ of a set of points is a sparse graph.

Corollary 12. *A realizable 2-factor F of a graph G is a subgraph of $G^{(2)}$.*

Proof: Let r_1, r_2, ..., r_n be radii that realize F. According to the proof of Lemma 11, we can assume that they are positive and smaller than or equal to $d^{(2)}(p_1)$, $d^{(2)}(p_2)$, ..., $d^{(2)}(p_n)$. Therefore, each edge $\{p_i, p_j\}$ belonging to F (we have $r_i + r_j \geq d_{ij}$) also belongs to $G^{(2)}$ (because $d^{(2)}(p_i) + d^{(2)}(p_j) \geq d_{ij}$). ■

2.1.2. Properties of the family of graphs $\mathbf{G}^{(2)}$

2.1.2.1. $\mathbf{G}^{(2)}$ of a Set of Points in the Plane is Sparse

In the previous section we have shown that a necklace tour is a subgraph of $G^{(2)}$. This section shows that, for each natural number m, the graph $G^{(m)}$ of a set of n points in the plane has at most $(19m-1)n$ edges. In particular, $G^{(2)}$ has at most $37n$ edges.

We need the following lemma.

Lemma 13. (Reifenberg [1948], Bateman and Erdős [1951]) *Let D_0 be a disk in the plane, and let S be a set of disks in the plane which are at least as big as D_0, such that none of their centers lies in the interior of D_0 or of another disk in S. Then the number of disks in S that intersect D_0 is at most 18.* ∎

With this lemma, we can prove the main result of this section which guarantees the sparsity of $G^{(m)}$.

Theorem 14. *Let P be a set of n points in the plane and let m be a natural number. Then the edge set $G^{(m)}$ of the graph $G^{(m)}$ of P contains at most $(19m-1)n$ edges.*

Proof: We claim that a smallest disk D_0 in $S^{(m)} = \{ D^{(m)}(p) \mid p \in P \}$ intersects at most $19m-1$ disks in $S^{(m)}$. From this the theorem follows by a simple inductive argument.

Let S be the set of disks in $S^{(m)}$ that intersect D_0. We know that each disk in S contains at most $m-1$ centers of other disks in its interior. We will now remove disks from S such that no center of a remaining disk lies in the interior of D_0 or of any other remaining disk. This is done by the following procedure:

Step 1: Remove all disks from S that have their centers in the interior of D_0.

Repeat the following step as long as there are disks in S which contain centers of other disks of S in their interior:

Step 2: Choose the largest among these disks and remove all disks from S whose centers belong to its interior.

Obviously, the final set S has the desired property and we conclude that it contains at most 18 disks.

In Step 1, at most $m-1$ disks are removed from S, since there are at most $m-1$ centers of disks in the interior of D_0. Whenever a disk is chosen in Step 2, that disk will not be chosen again. Moreover, the special choice as the largest disk guarantees that it will not be removed in a later step. Consequently, each disk which is chosen in Step 2 will be one of the disks which finally remain. It follows that Step 2 is repeated at most 18 times. Moreover, at most $m-1$ disks are removed from S in each repetition. Summing the the number of disks that are removed at each step and that finally remain, we get the following bound on the initial cardinality of the set S:

$$|S| \leq m - 1 + 18(m-1) + 18 = 19m - 1.$$ ∎

We conjecture that the worst-case configuration for the theorem consists of (approximately) equal-size disks. This would mean that the true upper bound for $\left|G^{(1)}\right|$ is $9n$ instead of $18n$, since all edges can be counted twice in the above proof. (Each disk is the smallest.) Consequently, we would obtain a bound of $(10m-1)n$ for $\left|G^{(m)}\right|$.

2.1.2.2. Construction of $\mathbf{G^{(2)}}$ for Points in the Plane

$G^{(2)}$ can be constructed using standard techniques from computational geometry.

Theorem 15. *The graph $G^{(2)}$ of a set P of n points in the plane can be computed in $O(n\log n)$ time and $O(n)$ space.*

Proof: In order to compute $G^{(2)}$ in $O(n\log n)$ time and $O(n)$ space, we first calculate the value $d^{(2)}(p)$ for each point p in P. For this purpose, we consider the 3^{rd}-order Voronoi diagram of P (see Preparata and Shamos [1985]). This is a partition of the plane into convex polygonal regions, where each region is the locus of points of the plane whose three nearest neighbors are some fixed three points. This partition can be constructed in $O(n\log n)$ time and $O(n)$ space (see Lee [1982]). Next, we compute the 2^{nd}-nearest neighbor of each of the n points in P in $O(n\log n)$ time. This is done by locating for each point p of P the region of the diagram it lies in (for a solution to this point location problem, see Edelsbrunner, Guibas, and Stolfi [1986]). Then we determine which of the three points given by the located region is furthest away from p. (One of the three points is p itself.) Thus, in $O(n\log n)$ time, we can determine the numbers $d^{(2)}(p)$, for all points p in P.

Next, we find all pairs $\{p_1,p_2\}$ of points in P whose corresponding disks intersect:

$$D^{(2)}(p_1)\cap D^{(2)}(p_2)\neq\emptyset.$$

By construction, no disk is contained in the interior of another disk. Hence, $D^{(2)}(p_1)\cap D^{(2)}(p_2)\neq\emptyset$ if and only if the bounding circles intersect. We thus have the problem of reporting all intersecting pairs of n circles. Using the plane-sweep technique of Bentley and Ottmann [1979], this can be done in $O(n\log n+k\log n)$ time, where k is the number of reported pairs. By Theorem 14, $k=O(n)$, and therefore the overall complexity is $O(n\log n)$. For all algorithms mentioned, $O(n)$ space is sufficient which concludes the proof of the theorem. ■

2.1.3. Testing the system of inequalities for feasibility

Systems of linear inequalities have become a well-established area in mathematics (cf. the more specific remarks in Section 2.1.3.2). Linear programming, which is just the minimization of a linear function subject to a system of linear inequalities, is by now a classical field of applied mathematics, although it is still a very active area of research, and many striking advances have been made only in the last few years.

In our case however, we do not want to optimize an objective function, we only want to check feasibility (although these two problems are theoretically equivalent, in a certain sense). Furthermore, our systems of inequalities have a special structure: Only two variables occur in each inequality, and the coefficients are ± 1. This will allow us to apply graph-theoretic methods for these systems.

In Section 2.1.3.1 we will transform the system of inequalities characterizing necklace tours into an equivalent system where all inequalities are of the form

$$v_j \leq v_i + c_{ij}, \text{ or}$$
$$v_j < v_i + c_{ij}.$$

Although this is necessary neither theoretically nor from a practical viewpoint, it will allow us a simpler formulation of the algorithms in the subsequent sections.

Section 2.1.3.2 discusses two possible approaches for solving these systems from a general point of view: the shortest path approach and the elimination approach.

A special class of elimination algorithms which are called generalized nested dissection require that the graphs can be separated into two approximately equal parts by removing a small subset of the vertices (a *separator*). Section 2.1.3.3 defines this property exactly and proves the separator theorem for our class of graphs. It is also shown how separators can be constructed.

Section 2.1.3.4 describes the generalized nested dissection method in detail. The conclusions from all these results are summarized in Section 2.3.

2.1.3.1. Transformation of the inequalities

The inequalities of the system (R.1) and (R.2$'$), whose solvability we want to check, are of the following two forms:

$$\text{(T)} \quad \begin{cases} r_i + r_j \geq d_{ij}, & \text{if } \{p_i, p_j\} \in F \qquad \text{(R.1)} \\ r_i + r_j < d_{ij}, & \text{if } \{p_i, p_j\} \in G^{(2)} \setminus F. \qquad \text{(R.2}'\text{)} \end{cases}$$

These inequalities are not suited to be modeled by a directed graph; therefore we transform them as follows: For the sake of symmetry, we first double the number of inequalities:

$$\text{(T}'\text{)} \quad \begin{cases} -r_i \leq r_j - d_{ij}, & \text{if } \{p_i, p_j\} \in F \\ -r_j \leq r_i - d_{ij}, & \text{if } \{p_i, p_j\} \in F \\ r_i < -r_j + d_{ij}, & \text{if } \{p_i, p_j\} \in G^{(2)} \setminus F \\ r_j < -r_i + d_{ij}, & \text{if } \{p_i, p_j\} \in G^{(2)} \setminus F. \end{cases}$$

In order to set up our directed graph, it is advantageous to introduce a new variable $\bar{r}_i$ for each r_i, where $\bar{r}_i$ represents $-r_i$.

Using the new variables, we rewrite system (T$'$) to

$$\text{(T}''\text{.1)} \quad \begin{cases} \bar{r}_i \leq r_j - d_{ij}, & \text{if } \{p_i, p_j\} \in F \\ \bar{r}_j \leq r_i - d_{ij}, & \text{if } \{p_i, p_j\} \in F \\ r_i < \bar{r}_j + d_{ij}, & \text{if } \{p_i, p_j\} \in G^{(2)} \setminus F \\ r_j < \bar{r}_i + d_{ij}, & \text{if } \{p_i, p_j\} \in G^{(2)} \setminus F. \end{cases}$$

With the additional constraints

$$\text{(T}''\text{.2)} \qquad r_i = -\bar{r}_i, \qquad \text{for all } i,$$

the system (T$''$) consisting of the constraints (T$''$.1) and (T$''$.2) is equivalent to the system (T$'$) and therefore to the system (T).

Lemma 16. *The system* (T$''$.1)-(T$''$.2) *has a solution if and only if the system* (T$''$.1) *has a solution.*

Proof: The "only if" part is clear. If $(r_i', \bar{r}_i')$ is a solution of (T$''$.1) then $(r_i, \bar{r}_i)$ with

$$r_i = \tfrac{1}{2}(r_i' - \bar{r}_i'), \text{ and } \bar{r}_i = \tfrac{1}{2}(\bar{r}_i' - r_i'),$$

is a solution of (T$''$.1) and (T$''$.2) as can be checked easily. ■

Hence, with respect to solvability, we can ignore the equations (T$''$.2), and we only have to consider inequalities of the form

$$v_j \leq v_i + c_{ij}, \quad \text{and}$$
$$v_j < v_i + c_{ij},$$

where the c_{ij} can be arbitrary (positive or negative) real numbers.

We remark that in general, the above transformation can be carried out for any system with at most two variables in each inequality and with coefficients ± 1. It can even be generalized to systems with arbitrary coefficients.

2.1.3.2. Graph-theoretic approaches to solving systems of inequalities

In the previous section, we have transformed the system of inequalities which we want to solve into an equivalent system where all inequalities are of the form

$$v_j \le v_i + c_{ij}, \quad \text{and}$$
$$v_j < v_i + c_{ij}.$$

For simplicity, let us first discuss systems with inequalities of the first form only:

$$v_j \le v_i + c_{ij}.$$

Let us draw a vertex for every variable v_j and an arc from v_i to v_j of weight c_{ij} for every inequality. We shall identify each variable with its corresponding vertex. Let us denote the resulting weighted directed graph by $G = (V, E, c)$.

It is well known that the solvability of the system of inequalities can be checked by a single-source shortest path computation on this graph (cf. Rockafellar [1984], Chapter 6). If the graph contains no negative cycles, the shortest path problem can be solved, and the shortest distances are a solution of the inequality system. If the graph contains a negative cycle, the system is unsolvable. This approach was followed in Edelsbrunner, Rote, and Welzl [1987]. The single-source shortest path problem in the presence of negative arc lengths can be solved by the Bellman-Ford algorithm in $O(|V|\,|E|)$ time (see Lawler [1976]). Since $|V| = O(n)$ and $|E| = O(n)$ in our case, this leads to an $O(n^2)$ algorithm for checking the system.

One only has to take special care of the strict inequalities: If the graph contains a cycle of zero length which contains an arc corresponding to a strict inequality, the system is also infeasible. The checking of this additional condition and the modification of the shortest distances to account for the strict inequalities are described in detail in Edelsbrunner, Rote, and Welzl [1987], Section 5.

On the other hand, we can use elimination procedures for solving systems of inequalities of the type considered. Although elimination schemes like Gaußian elimination or Gauß-Jordan elimination were first applied in numerical linear algebra and were hence originally described in matrix terminology, they can be conveniently described in a graph-theoretic framework, which is also more appropriate in the context of our application.

Let us again consider the graph $G = (V, E, c)$ corresponding to a system of weak inequalities of the form

$$v_j \le v_i + c_{ij}.$$

The above system can be expressed as

$$\max_{(j,k)\in E} (v_k - c_{jk}) \le v_j \le \min_{(i,j)\in E} (v_i + c_{ij}), \qquad \text{for all } j.$$

Let us look at all arcs that enter or leave a vertex v_j (Figure 12a). We can eliminate the variable v_j and its incident arcs by inserting direct arcs from all its predecessors to all its successors. More specifically, for each pair of incoming and outgoing arcs, we add

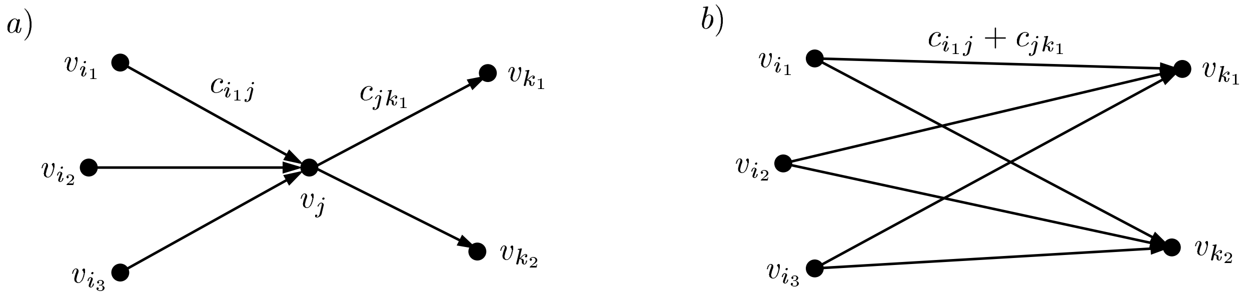

Figure 12. Elimination of a vertex.

a direct arc whose weight is the sum of the weights of the two arcs (cf. Figure 12b). (If some vertex is both a predecessor and a successor of v_j, a loop is added to it.) The time required for this elimination operation is proportional to the number of new arcs, which is the product of the indegree and the outdegree of v_j.

After having solved the resulting system without variable v_j, we can give v_j any value between

$$\max_{(j,k)\in E}(v_k - c_{jk}) \qquad \text{and} \qquad \min_{(i,j)\in E}(v_i + c_{ij}).$$

(If there is no incoming or no outgoing arc, the expression is taken to be $-\infty$ or $+\infty$, resp.) The arcs that were inserted in the elimination step ensure that this range is non-empty. This step is called "back-substitution".

As soon as we detect a loop with negative weight, there is obviously no solution to the system, since a loop means

$$v_i \leq v_i + c_{ii};$$

otherwise, we can continue until we have eliminated all but one variable and we are left with a single-vertex graph. If it has no negative loop, we may give any value to this variable and start back-substituting the other variables in the reverse order in which they were eliminated.

The modifications of this algorithm which are required in order to handle strict inequalities of the form

$$v_j < v_i + c_{ij}$$

are straightforward: During elimination, an inserted arc becomes strict if either of the two arcs from which it originated was strict. A loop of weight zero will terminate the algorithm if it is connected with a strict inequality. And finally, during backsubstitution, the endpoints of the interval of possible values may have to be excluded.

Pairwise elimination schemes like the one above, but for general systems of inequalities, have already been suggested by Fourier [1827] (see also Duffin [1974]). These algorithms, commonly known as Fourier-Motzkin elimination, are not polynomial in general. Systems in which at most two variables occur in each inequality have attracted special attention, since they allow us to impose a directed graph structure on the problem and to utilize this structure for a more efficient solution. In particular, Megiddo [1983], based on a technique in Shostak [1981], gives an $O(m \cdot n^3 \log m)$ time algorithm to solve systems of m inequalities and n variables, where each inequality contains at most two variables.

The connection between algorithms for solving systems of linear equations and algorithms for certain graph problems has been known for some time (cf. e. g. Carré [1971]). In the meantime, a common algebraic foundation for both types of problems (and others) has been established (cf. Lehmann [1977], Carré [1979], Gondran and Minoux [1979], Zimmermann [1981] Chapter 8). In order to formulate them in the proper algebraic setting one has to use a semiring. The linear algebra problems use the semiring $(\mathbb{R}, +, \cdot)$, which is even a ring, whereas the shortest path problems use the semiring $(\mathbb{R}, \max, +)$. The algorithm presented above corresponds to Gaußian elimination. Conventionally, the elimination algorithms are described algebraically. In our application, we can directly describe the idea of vertex elimination using graph-theoretic terms.

Of course, the amount of work and storage space depends on the number of new arcs which are inserted during the elimination process (the so-called "fill-in"), and this number again depends on the elimination ordering. In the worst case, conventional Gaußian elimination (for dense matrices) takes $O(n^3)$ steps and $O(n^2)$ space. However, by using the special structure of the underlying graph, we can reduce the time to $O(n^{3/2})$ and the space to $O(n)$, even without taking advantage of the property that all non-zero coefficients have absolute value 1.

In the next section we will define the property that is required of the graphs, and we will show that it holds for the graphs $G^{(2)}$ which are of interest to us. Afterwards we return to the topic of vertex elimination algorithms, and we describe how the time complexity claimed above can be achieved.

2.1.3.3. $\sqrt{\mathbf{n}}$-separators for $\mathbf{G}^{(2)}$

Let us first define what we mean by a separator theorem for a family of graphs. Let a class of graphs be given which is closed under the subgraph relation, i. e., any subgraph of a graph of this class belongs also to this class. We say that this class of graphs satisfies an *$f(n)$-separator theorem* if there are constants $\alpha < 1$ and $\beta > 0$ such that the following holds: If a graph in this class has n vertices, then the vertex set can be partitioned into three sets A, B, and C such that $|A|, |B| \leq \alpha n$, $|C| \leq \beta f(n)$, and no vertex in A is adjacent to a vertex in B. C is called a *separator* of this graph.

This definition does not apply to individual graphs; it can only apply to a whole family of graphs because one wants to apply the separator theorem recursively to the parts A and B. Therefore it is important that the family is closed under the subgraph relation. (A graph might have an articulation point, i. e., an $O(1)$-separator splitting the graph into to equal-sized parts, and nevertheless these two parts might not have good separators.) It is also important that the two "halves" A and B are of approximately equal size; this guarantees that the recursive ("nested") dissection will terminate at a recursive depth of at most $O(\log n)$.

In this section we prove the $\sqrt{n}$-separator theorem for the family of graphs $G^{(2)}$, and we show how the separators can be constructed. The family of graphs $G^{(2)}$ is not itself closed under the subgraph relation; therefore we have to mention their subgraphs explicitly when we formulate the theorem.

Our results are based on the corresponding results for planar graphs, and hence we cite these results first. They use a generalization of the concept of separators, where the vertices have weights, and the size of the parts A and B is measured by their total weights. (The size of the separator is still measured by its cardinality.)

Theorem 17 (weighted $\sqrt{n}$-separator theorem for planar graphs). (Lipton and Tarjan [1979]). *Let G be a planar graph with n vertices having non-negative vertex weights summing to no more than 1. Then the vertices of G can be partitioned into three sets A, B, and C, such that no edge joins a vertex in A with a vertex in B, the total weight of both A and of B is at most $2/3$, and $|C| \le \sqrt{8n}$.*
Moreover, the set C can be constructed in $O(n)$ time and space if the adjacent vertices of each vertex are given in clockwise order (for some planar embedding of the graph). ■

Theorem 18. *Let G be the graph $G^{(m)}$ of a set of n points in the plane or a subgraph of such a graph. Then its vertex set can be partitioned into three parts A, B, and C, such that the following properties hold: there are no edges between vertices of A and vertices of B; C contains at most $8\sqrt{(19m-1)}\sqrt{n}$ vertices; and A and B consist of at most $2n/3$ vertices each.*

Proof: Let P_0 be the set of points from which G originated (possibly as a subgraph). It is sufficient to show the theorem for the graph $G^{(m)}$ of the set P, where P is the subset of points which are vertices of G. We assume $n > m$, for otherwise $C = P$, $A = B = \emptyset$ would fulfill the requirements of the theorem.

We would like to apply the $\sqrt{n}$-separator theorem for planar graphs. However, the graph $G^{(m)}$, although defined through geometrical relations in the plane, is itself not planar. Thus we have to find a planar graph which models the adjacencies of $G^{(m)}$. We use a graph which is defined from the geometrical layout of the circles which define $G^{(m)}$.

Consider the set of circles of radii $d^{(m)}(p)$ which define $G^{(m)}$. (By a *circle*, we mean only the boundary, as opposed to a *disk*.) We want to exclude touching circles or intersections of more than two circles in a single point from consideration. Therefore we increase the radius of each circle by a positive amount ε which is small enough such that no new intersections of circles appear, but which ensures that no two circles touch and no three circles intersect (cf. Figure 13a and b). The case that one circle touches another from inside (Figure 13c), where an increase of both radii by the same amount would not change anything, cannot occur since no circle can contain another. (Otherwise it could be handled by enlarging smaller circles more than larger ones.)

Now we denote by K the (multi-)graph whose vertices are the intersections of the circles, and whose edges correspond to the circular arcs between these points (see Figure 14).

This graph is planar. Each vertex has degree 4. For each pair of intersecting circles, K contains exactly 2 vertices. Hence K has twice as many vertices as $G^{(m)}$ has edges. Each circle intersects at least one other circle, and hence it contains at least two points. We now attach weights to the vertices of K as follows: We start with all weights equal to 0.

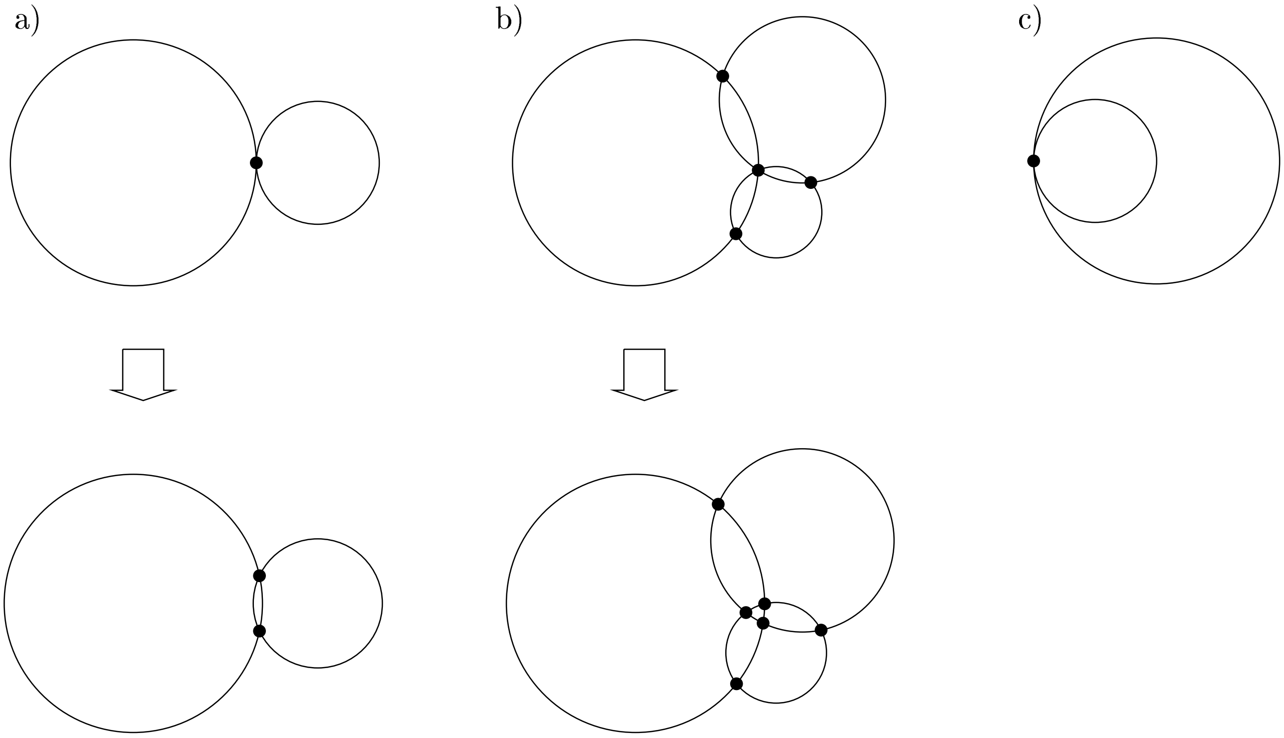

Figure 13. How to handle degenerate situations

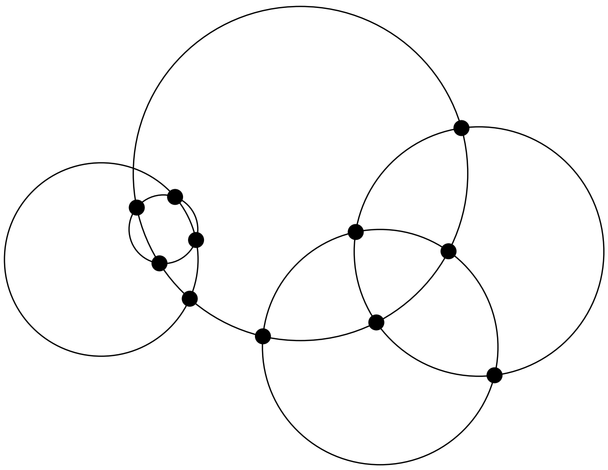

Figure 14. The graph K

For each point p of P we select an arbitrary vertex $r(p)$ on the circle corresponding to p, and add $1/n$ to its weight. Thus the total weight becomes 1, and each vertex has weight 0, $1/n$, or $2/n$.

Now we apply the weighted planar separator theorem to K and we obtain three sets A_K, B_K, and C_K, where C_K separates A_K and B_K.

We claim that the set C of points of P whose circles contain the points of C_K is an $O(\sqrt{n})$-separator for the graph $G^{(m)}$. Since each point of K belongs to two circles, $|C| \leq 2\,|C_K|$.

The sets A and B are defined as the sets of points p of P which do not belong to C and whose representative points $r(p)$ belong to A_K and B_K, resp.

We have to prove that $G^{(m)}$ contains no edge between A and B. If two points p and q of P are connected by an edge in $G^{(m)}$ then the two circles corresponding to p and q intersect. The two vertices $r(p)$ and $r(q)$ which carry the weights of p and q must be connected by a path in K which uses only edges which are pieces of the two circles.

If neither p not q belongs to C then this path must still exist in the graph $K \setminus C_K$. Hence $r(p)$ and $r(q)$ belong to the same component of $K \setminus C_K$, i. e., they belong either both to A_K or both to B_K. Concluding, we have proved that if neither p nor q belongs to C, then they belong both to A or both to B.

We still have to prove the claimed bounds on the cardinalities of A, B, and C. A_K and B_K have weight at most $2/3$, hence A and B contain at most $2n/3$ points. C_K contains at most $\sqrt{8}\sqrt{|V(K)|} = \sqrt{8}\sqrt{2\left|G^{(m)}\right|}$ vertices. Since $\left|G^{(m)}\right| \leq (19m-1)n$, we have $|C_K| \leq \sqrt{16(19m-1)n}$, and hence $|C| \leq 2\,|C_K| \leq 8\sqrt{(19m-1)n}$. ■

Corollary 19. *The family of the graphs $G^{(2)}$ and their subgraphs satisfies a $\sqrt{n}$-separator theorem with $\alpha = 2/3$ and $\beta = 8\sqrt{37}$.* ■

Finally, we address the question how the separator for a graph $G^{(2)}$ can be found quickly. It is clear from the proof of the preceding theorem that if the graph K is available, the rest of the construction needs only linear time: Since each vertex of K has only degree 4, the requirement of Theorem 17 that the neighbors are given in sorted order can be trivially fulfilled in linear time.

The following theorem shows that, for a given graph $G^{(2)}$, K can be constructed in $O(n \log n)$ time. In Section 2.1.4., where separators have to be found repeatedly for different subgraphs of $G^{(2)}$, we will build the graphs K for these subgraphs as kind of "subgraphs" of the graph K of the original graph $G^{(2)}$. It will turn out that in the context of that application each individual separator construction takes only linear time.

Lemma 20. *The graph K of the previous theorem can be constructed in $O(n \log n)$ time, along with the construction of $G^{(2)}$.*

We refer to the construction of $G^{(2)}$ according to Theorem 15. There, all intersections of circles were computed. These intersections form the vertex set of K, and the edges of K, which are arcs of the circles, are readily available during the plane-sweep. It only remains to discuss how the degeneracies, namely two circles which touch, and three of more circles which intersect in a common point, should be treated. Touching circles can be handled by simply generating two vertices instead of one. If three or more circles intersect in one point, the plane-sweep algorithm will notice this as two intersections which have the same coordinates. Since it processes the intersections according to the value of one coordinate, this means that there is a tie in deciding which intersection to process next. The algorithm can break this tie by implicitly applying the trick of enlarging the circles by a small amount ε, as in the proof of the previous theorem. In this way, the intersections, which are actually one point, will appear as different intersections, and they will yield different vertices of the graph K. ■

2.1.3.4. Generalized nested dissection for graphs with $\sqrt{n}$-separators

Generalized nested dissection is described in detail in Lipton, Rose, and Tarjan [1979]. They showed that for families of graphs fulfilling an n^α-separator theorem (where $0 \leq \alpha < 1$), generalized nested dissection provides an asymptotically optimal elimination ordering.

However, for $\sqrt{n}$-separators, their version uses $\Omega(n \log n)$ space, and they give only a reference to a paper where it is shown in the case of (specialized) nested dissection (cf. George [1973]) for two-dimensional rectangular grids, how a storage reduction to $O(n)$ can be achieved (cf. Eisenstat, Schultz, and Sherman [1976], Section 4, pp. 91–95). Therefore we give a self-contained presentation of the storage-saving version of the generalized nested dissection algorithm. We present it in a recursive fashion, whereas in Lipton, Rose, and Tarjan [1979] it is described as establishing an elimination ordering. (This elimination ordering is nothing but a post-order traversal of the tree which is built up in the recursive algorithm.)

2.1.3.4.1. Generalized nested dissection without storage saving

Generalized nested dissection can be regarded as an application of the divide-and-conquer paradigm. The idea is roughly the following:

We apply the separator theorem to the given graph and obtain the sets A, B, and C, where C separates A from B (cf. Figure 15a). Then we first eliminate A and B, and finally C; since there are no arcs between A and B, the two eliminations of A and B are completely independent of each other. When eliminating A and B, we recursively use the same procedure.

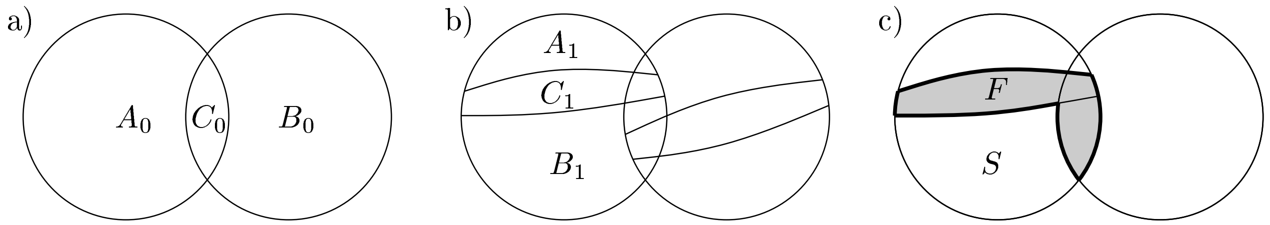

Figure 15. A global view of generalized nested dissection.

Since we recursively dissect the vertex set into smaller and smaller pieces, we can model the result of these dissections on a tree: We proceed from the root to the leaves when building the dissection tree; we proceed from the leaves to the root during the elimination phase; and we again proceed from the root to the leaves during back-substitution. Since we walk through the tree several times, it is best to store the tree explicitly.

A more formal description of the procedure follows:

Phase 1: Construction of the tree

We assume the existence of a procedure

dissect(V, A, B, C);

taking a vertex set V as argument, and partitioning it into three sets A, B, and C according to the separator theorem applied to the subgraph induced by V. The following procedure *dissection_tree* recursively dissects the vertex set and builds the corresponding dissection tree. When the vertex set becomes smaller than some cut-off value N_0, the dissection stops. N_0 is determined from the constants in the separator theorem such that it is guaranteed that the partition does not become the trivial partition $C = V$, $A = B = \emptyset$. The procedure returns a pointer to a node *tree_node* which contains three fields: A field *vertices* storing a set of vertices and two pointers to the *left* and *right* subtrees.

```
function dissection_tree(S, F: vertex_set) returns tree_node;
begin let x be a new tree_node;
        if |S ∪ F| < N0
           then begin vertices(x) := S;
                      left(x) := right(x) := null;
                end
           else begin dissect(S ∪ F, A, B, C);
                      vertices(x) := C \ F;
                      left(x)   := dissection_tree(A \ F, (F ∩ A) ∪ C);
                      right(x)  := dissection_tree(B \ F, (F ∩ B) ∪ C);
                end;
        return x;
end;
```

The procedure *dissection_tree.*

When the first dissection into three sets A_0, B_0, and C_0 has been found, C_0 is stored in the root of the tree, and A_0 and B_0 will go into the left and right subtrees, resp. The next sets to be separated are $A_0 \cup C_0$ and $B_0 \cup C_0$ (see Figure 15b). When $A_0 \cup C_0$ has been partitioned into A_1, B_1, and C_1, $C_1 \setminus C_0$ is stored as the vertex set of the left son of the root. Only vertices which have not been stored already in some ancestor of the current node are stored in a node. In this way, since each vertex is stored exactly once, the tree takes linear space.

In general, the vertex sets to which the separator theorem is applied will contain, in addition to the set A_j (or B_j) that has been separated from the rest of the graph by a sequence of separators, all vertices to which arcs from vertices in A_j might lead; these vertices are contained in the sets C_i that were used as separators in the path from the root to the current node in the tree. The procedure *dissection_tree* takes two arguments: the first one — S — is the set of vertices which have been separated from the rest of the graph, and the second one — F — is a set of vertices which separate S from the rest of the graph: there are no arcs between S and $V \setminus (S \cup F)$ (cf. Figure 15c). $S \cup F$ is the vertex set to be separated next. Although the sets S and F are not stored in the nodes we will later refer to the current cardinality of these parameters at the time of the call of *dissection_tree* by which a specific node was generated.

The procedure returns a pointer to a node of the tree; it is initially called as follows:

$$root := dissection\_tree(V, \emptyset).$$

Lipton, Rose, and Tarjan [1979] proved that this call takes $O(n \log n)$ time if each call to *dissect* takes linear time. We will show below in Section 2.1.3.4.4 that we can also achieve this time bound in our case.

Phase 2: Elimination

Elimination of all nodes can be done in a post-order traversal of the tree. Starting at the leaves, the vertices stored in the nodes of the tree are successively eliminated. We initiate elimination by calling

$$eliminate\_all(root).$$

```
procedure eliminate_all(x: tree_node);
begin if x ≠ null
        then begin eliminate_all(left(x));
                   eliminate_all(right(x));
                   for all v ∈ vertices(x)
                       do eliminate_vertex(v);
             end;
end;
```

The procedure *eliminate_all.*

Phase 3: Back-substitution

Back-substitution is done in a preorder traversal of the tree: The vertices in each node of the tree are back-substituted (i. e., the values of the corresponding variables are computed as described in Section 2.1.3.2) before the nodes in the subtrees are visited (where these values are already needed.) As above, we start the back-substitution process by calling

$$back\_substitute\_all(root).$$

```
procedure back_substitute_all(x: tree_node);
begin if x ≠ null
        then begin for all v ∈ vertices(x) (* in the reverse order in which *)
                                            (* they were eliminated         *)
                       do back_substitute_vertex(v);
                   back_substitute_all(left(x));
                   back_substitute_all(right(x));
             end;
end;
```

The procedure *back_substitute_all.*

2.1.3.4.2. Analysis of the fill-in

Let us analyze the number of arcs that are inserted by the elimination. First, an inserted arc that starts in a vertex stored in some node of the tree can only go to a vertex stored in the same node or in an ancestor or a descendant of this node; there can be no "cross arcs" relative to the tree: When *eliminate_all*(x) is called for some node x, there are initially no arcs between the vertices stored in the subtree *left*(x) and the vertices stored in the subtree *right*(x). When eliminating these vertices, new arcs are inserted only between vertices that are adjacent to the same node, i. e., between pairs of vertices which are both stored in *left*(x) or in *right*(x), or between such vertices and vertices stored in x.

In the following treatment we closely follow the analysis given in Lipton, Rose, and Tarjan [1979].

For estimating the number of inserted arcs, we first count the number of arcs between vertices stored in the same node x. If x was built in a call to *dissection_tree* with parameters S and F and the dissection of $S \cup F$ was $A \cup B \cup C$, then $|C|$ is an upper bound on the number of vertices stored in x, and hence $|C|^2$ is an upper bound on the number of arcs between vertices stored in x.

Now let us count the number of arcs that run between a vertex stored in x and a vertex stored in an ancestor of x (in either direction). F is the set of vertices stored in ancestors of x to which the vertices stored in x (and in the whole subtree rooted at x, i. e., the vertices in S) can possibly be adjacent. Thus, $2\,|C|\,|F|$ is an upper bound on the number of these arcs.

In the following, we derive an upper bound $f(n,l)$ on the number of inserted arcs which are incident to vertices stored in the subtree rooted at a node x which was created with $|S \cup F| = n$, $|F| = l$ and $|C| = c$. We have

$$\begin{aligned} f(n,l) &= c^2 + 2cl && \text{(for the vertices in } x\text{)} \\ &\quad + \max_{n_1,n_2,l_1,l_2} \left(f(n_1,l_1) + f(n_2,l_2)\right) && \text{(for the vertices in the subtrees)} \end{aligned} \tag{7}$$

Let us find out over which values the maximum in this expression has to be taken: n_1, l_1 and n_2, l_2 are the values of $|S \cup F|$ and $|F|$ for the calls to *dissection_tree* for *left*(x) and *right*(x), resp., i. e.,

$$\begin{aligned} n_1 &= |A \cup C| \qquad \text{and} \qquad l_1 = |(F \cap A) \cup C|\,, \\ n_2 &= |B \cup C| \qquad \text{and} \qquad l_2 = |(F \cap B) \cup C|\,. \end{aligned}$$

We have

$$\begin{aligned} n_1 &= |A \cup C| \le |A| + |C| \le 2n/3 + c, \text{ and} \\ n_1 &= |A \cup C| = |A \cup B \cup C| - |B| \ge n - 2n/3 = n/3, \end{aligned}$$

and analogous inequalities for n_2; thus,

$$n/3 \le n_1, n_2 \le 2n/3 + c.$$

Moreover,

$$\begin{aligned} n_1 + n_2 &= |A \cup C| + |B \cup C| = |A \cup B \cup C| + |C| = n + c, \text{ and} \\ l_1 + l_2 &= |(F \cap A) \cup C| + |(F \cap B) \cup C| \\ &\le |F \cap A| + |C| + |F \cap B| + |C| \le |F| + 2\,|C| = l + 2c. \end{aligned}$$

By our separator theorem (Corollary 19), we have $c \le \beta\sqrt{n}$, and we can write

$$f(n,l) = \begin{cases} n^2, & \text{for } n \le N_0, \\ \beta^2 n + 2\beta l\sqrt{n} + \max\limits_{\substack{n/3 \le n_1, n_2 \le 2n/3+\beta\sqrt{n} \\ n_1+n_2 \le n+\beta\sqrt{n} \\ l_1+l_2 \le l+2\beta\sqrt{n}}} \left(f(n_1,l_1) + f(n_2,l_2)\right), & \text{for } n > N_0. \end{cases}$$

By observing that $f(n,l)$ is increasing in l, we can get a lower bound for $f(n,l)$ if we replace l by 0 on the right side and set $n_1 = n_2 = n/2$:

$$f(n,0) \ge \beta^2 n + 2f(n/2,0).$$

From this recursion follows easily that $f(n,0)$, the upper bound for the total number of inserted arcs in the algorithm, is $\Omega(n\log n)$. (It is also known that the number of inserted arcs can really become as large as $\Omega(n\log n)$. Lipton, Rose, and Tarjan [1979] show that the maximum number is actually $\Theta(n\log n)$.)

2.1.3.4.3. Reducing the storage requirement

How can this superlinear space requirement be reduced? From the above recursion we see that we must reduce the factor 2 on the right side; i. e., instead of needing the storage for two subproblems of approximately half the size we must require only the storage for one solution. This can be done by freeing the storage used by the first call to *eliminate_all*(*left*(x)) before the second call *eliminate_all*(*right*(x)), i. e., globally speaking, by selectively forgetting and recomputing arcs. This will increase the computation time by only a constant factor.

Let us discuss the required modifications in detail. Assume that after returning from the call to *eliminate_all*(x) all arcs incident to vertices stored in x are deleted. Then, after calling *eliminate_all*(*left*(*root*)) and *eliminate_all*(*right*(*root*)), we could still eliminate the variables in *vertex*(*root*) and back-substitute them. However, we could not compute the values of the other variables, since the necessary arcs were deleted. We have to repeat the elimination process, first for *left*(*root*), then for *right*(*root*). However, this time we now do not throw away the arcs incident to *vertex*(*left*(*root*)), but we use them to back-substitute their values.

Thus, there are now two versions of *eliminate_all*: one with deletion, and one with backsubstitution; both are variations of the original procedure.

The whole computation is started by calling

eliminate_back_substitute(*root*).

It can be proved by induction that

(i) after the procedure *eliminate_delete*(x) or *eliminate_back_substitute*(x) returns to the caller the graph is restored to its original status before the procedure was called;

(ii) when either *eliminate_delete*(x) or *eliminate_back_substitute*(x) is being called, the only inserted arcs run between vertices that are stored in nodes on the path from the *root* to x.

```
procedure eliminate_delete(x: tree_node);
begin if x ≠ null
        then begin eliminate_delete(left(x));
                   eliminate_delete(right(x));
                   for all v ∈ vertices(x)
                       do begin eliminate_vertex(v);
                                delete all inserted arcs incident to v;
                          end;
             end;
end;
```

The procedure *eliminate_delete.*

```
procedure eliminate_back_substitute(x: tree_node);
begin if x ≠ null
        then begin eliminate_delete(left(x));
                   eliminate_delete(right(x));
                   for all v ∈ vertices(x)
                       do eliminate_vertex(v);
                   for all v ∈ vertices(x) (* in the reverse order as above *)
                       do back_substitute_vertex(v);
                   eliminate_back_substitute(left(x));
                   eliminate_back_substitute(right(x));
                   for v ∈ vertices(x)
                       do delete all inserted arcs incident to v;
             end;
end;
```

The procedure *eliminate_back_substitute.*

Both properties follow from the fact that in the local statements of both procedures (not during the execution of recursive calls) no arcs are affected except those that run between vertices in x and vertices that are stored in nodes on the path from the *root* to x.

2.1.3.4.4. Performance analysis of the complete algorithm

Let us now analyze the storage requirement and the computation time required by the modified procedure.

We define $\hat{f}(n,l)$ to be an upper bound on the storage space needed for either *eliminate_delete* or *eliminate_back_substitute*, where the same conventions hold with respect to the meaning of n and l as with $f(n,l)$.

The recursion for $\hat{f}(n,l)$ is also similar to the recursion (7) for $f(n,l)$, except that the sum of the requirements for the two subproblems has to be replaced by their maximum, by property (i) above:

$$\hat{f}(n,l) = \begin{cases} n^2, & \text{for } n \le N_0, \\ \beta^2 n + 2\beta l\sqrt{n} + \max\limits_{\substack{n/3 \le n_1, n_2 \le 2n/3+\beta\sqrt{n} \\ n_1+n_2 \le n+\beta\sqrt{n} \\ l_1+l_2 \le l+2\beta\sqrt{n}}} \max\left\{\hat{f}(n_1,l_1), \hat{f}(n_2,l_2)\right\}, & \text{for } n > N_0. \end{cases}$$

Since $\hat{f}(n,l)$ is isotone in both n and l, we can simplify this expression:

$$\hat{f}(n,l) \le \beta^2 n + 2\beta l\sqrt{n} + \hat{f}(2n/3+\beta\sqrt{n},\ l+2\beta\sqrt{n}), \qquad \text{for } n > N_0.$$

We shall prove that $\hat{f}(n) = O(n)$.

Let $N_1 \ge N_0$ be a number such that

$$2n/3 + \beta\sqrt{n} \le 3n/4, \qquad \text{for } n > N_1.$$

Then we can write

$$\hat{f}(n,l) \le \begin{cases} n^2, & \text{for } n \le N_1, \\ \beta^2 n + 2\beta l\sqrt{n} + \hat{f}(3n/4,\ l+2\beta\sqrt{n}), & \text{for } n > N_1. \end{cases}$$

Denoting the additive term in the recursion by

$$h(n,l) = \beta^2 n + 2\beta l\sqrt{n}$$

and expanding the recursion, we get

$$\hat{f}(n,0) \le h(n,0) + h(n_1,l_1) + h(n_2,l_2) + \cdots + N_1^2,$$

where

$$n_{i+1} = (3/4)n_i = n\,(3/4)^i \quad \text{and} \quad l_{i+1} = l_i + \beta\sqrt{n_i}, \quad l_0 = 0.$$

We get

$$\begin{aligned} l_i &= 0 + \beta\sqrt{n} + \beta\sqrt{n_1} + \cdots \beta\sqrt{n_{i-1}} \\ &\le \beta\sqrt{n}\left(1 + \sqrt{3/4} + \sqrt{3/4}^2 + \sqrt{3/4}^3 + \cdots\right) = \beta\gamma\sqrt{n}, \end{aligned} \tag{8}$$

with $\gamma = 1/(1-\sqrt{3/4}) = 7.4641$. Thus,

$$\begin{aligned}
\hat{f}(n,0) &\le N_1^2 + \sum_{i=0}^{\infty} \left(\beta^2 n_i + 2\beta l_i \sqrt{n_i}\right) \\
&\le N_1^2 + \sum_{i=0}^{\infty} \left(\beta^2 n_i + 2\beta^2 \gamma n_i\right) \\
&= N_1^2 + \beta^2(1+2\gamma)n \sum_{i=0}^{\infty} (3/4)^i \\
&= N_1^2 + 4\beta^2(1+2\gamma)n = O(n).
\end{aligned}$$

Thus we have proved:

Theorem 21. *The elimination step in the modified algorithm takes $O(n)$ space.* ■

Remark: Actually, we have only proved that the amount of fill-in that has to be stored is $O(n)$. The theorem is further supported by the implementational details which are given in the next subsection.

Let us now analyze the running time of our algorithm. For the time complexity of *eliminate_delete*, as well as of *eliminate_all*, we can set up a recursion as follows: The amount of work for the elimination of a single vertex is proportional to the product of the indegree and the outdegree of this vertex. In a call to any of the elimination procedures, the indegree and the outdegree are bounded by $c+l$, where c and l have the same meaning as in Section 2.1.3.4.2. Since c vertices are eliminated, the time spent inside the subroutine, exclusive of the recursive calls, is $O(c(c+l)^2)$. Hence the total time requirement for *eliminate_delete* or for *eliminate_all*, which we denote by $g(n,l)$, satisfies the following recursion

$$\begin{aligned}
g(n,l) &= c(c+l)^2 + \max_{n_1,n_2,l_1,l_2} \left(g(n_1,l_1) + g(n_2,l_2)\right) \\
&\le \beta\sqrt{n}(\beta\sqrt{n}+l)^2 + \max_{n_1,n_2,l_1,l_2} \left(g(n_1,l_1) + g(n_2,l_2)\right) \\
&= O(\sqrt{n}(\sqrt{n}+l)^2) + \max_{n_1,n_2,l_1,l_2} \left(g(n_1,l_1) + g(n_2,l_2)\right),
\end{aligned}$$

where the maximum has to be taken over the same range as for the recursion (7) in Section 2.1.3.4.2, and suitable initial conditions for $n \le N_0$ have to be provided. Lipton, Rose, and Tarjan [1979] (p. 350–351, proof of Theorem 3) have shown that this recursion has the solution

$$g(n,l) = O(\sqrt{n}(\sqrt{n}+l)^2),$$

and hence

$$g(n,0) = O(n^{3/2}).$$

Now let us denote by $\hat{g}(n,l)$ the time for the procedure *eliminate_back_substitute*. The recursion for $\hat{g}(n,l)$ is similar to the above recursion for $g(n,l)$, except that we have to add $2g(n,l)$ on the right side for the two calls to *eliminate_delete*. (The time for backsubstitution of a vertex is only the sum of the indegree and the outdegree of the vertex, hence it is certainly dominated by the time for elimation and can be ignored in the asymptotic complexity.)

$$\begin{aligned}\hat{g}(n,l) &= c(c+l)^2 + 2g(n,l) + \max_{n_1,n_2,l_1,l_2} (\hat{g}(n_1,l_1) + \hat{g}(n_2,l_2)) \\ &= O(\sqrt{n}(\sqrt{n}+l)^2) + 2g(n,l) + \max_{n_1,n_2,l_1,l_2} (\hat{g}(n_1,l_1) + \hat{g}(n_2,l_2))\,,\end{aligned}$$

However, it was shown above that $g(n,l) = O(\sqrt{n}(\sqrt{n}+l)^2)$. The term $2g(n,l)$ can therefore be subsumed under the O-expression, and we get essentially the same recursion as above, only with a different constant:

$$\hat{g}(n,l) = O(\sqrt{n}(\sqrt{n}+l)^2) + \max_{n_1,n_2,l_1,l_2} (\hat{g}(n_1,l_1) + \hat{g}(n_2,l_2))\,.$$

From this we conclude as above:

$$\hat{g}(n,0) = O(n^{3/2}).$$

Thus we have proved:

Theorem 22. *The elimination step in the modified algorithm takes $O(n^{3/2})$ time.* ■

The only part of the generalized nested dissection algorithm that has not been analyzed yet is the construction of the dissection tree. We will show that both the work and the storage at the local level of a call to *dissection_tree* with $|S \cup F| = n$ are $O(n)$.

We assume that at the initial level we are given the graphs $G^{(2)}$ and K. Then the separator C can be constructed in linear time, as follows from the discussion in Section 2.1.3.3. The only problem is to ensure that the appropriate subgraphs of $G^{(2)}$ and the corresponding graphs K are available for the recursive calls, such that during these calls the separators can also be found in linear time. In other words, we are given a subgraph of $G^{(2)}$ induced by the vertex set $S \cup F$ and the corresponding graph K; after finding the separator C and the sets A and B, we have to construct the graph induced by $A \cup C$ and the graph K corresponding to it before the first recursive call, and similarly for the second recursive call.

It is straightforward to remove all edges incident to B in order to obtain the subgraph of $G^{(2)}$ induced by $A \cup C$. We update the graph K as follows: For each point of B, a whole circle of the graph K must disappear. For each circle, we go through the vertices on that circle one by one. Each vertex which is encountered lies on two circles: the circle which is about to disappear, and another circle. When the vertex is removed, its neighbors on this other circle become adjacent, and the graph K must be updated to reflect this change. It is clear that this can be done in time proportional to the size of the graph which is $O(n)$.

Before the change, we make a local copy of both graphs that we deal with; so we do not have to worry about restoring them for the second recursive call. Thus we need $O(n)$ time and space locally for each call. This leads to the following recursions for the total time $h(n,l)$ and the total space $q(n,l)$ required by the procedure *dissection_tree*:

$$h(n,l) = O(n) + \max_{\substack{n/3 \le n_1,n_2 \le 2n/3+\beta\sqrt{n} \\ n_1+n_2 \le n+\beta\sqrt{n} \\ l_1+l_2 \le l+2\beta\sqrt{n}}} (h(n_1,l_1) + h(n_2,l_2)), \qquad \text{for } n > N_0,$$

and

$$q(n,l) = O(n) + \max_{\substack{n/3 \le n_1, n_2 \le 2n/3 + \beta\sqrt{n} \\ n_1 + n_2 \le n + \beta\sqrt{n} \\ l_1 + l_2 \le l + 2\beta\sqrt{n}}} \max\{q(n_1,l_1), q(n_2,l_2)\}, \qquad \text{for } n > N_0.$$

For the time complexity $h(n,l)$ we can refer to Theorem 2 of Lipton, Rose, and Tarjan [1979], where

$$h(n,0) = O(n \log n)$$

is obtained.

For $q(n,l)$ we use the same trick as for the derivation of $\hat{f}(n,l)$, and we obtain

$$q(n,l) = O(n) + q(3n/4,\ l + 2\beta\sqrt{n}), \quad \text{for } n > N_1.$$

We can proceed in the same way as for $\hat{f}(n,l)$ and we get

$$q(n,0) = O(n).$$

Summarizing, we have

Theorem 23. *The dissection tree can be built in $O(n \log n)$ time and $O(n)$ space.* ■

2.1.3.4.5. Details about the implementation

Since we talk about forgetting (i. e., deleting) arcs, we have to become specific about the way in which the graph is stored.

As we are dealing with sparse graphs, an adjacency list representation will probably be appropriate. However, during elimination, arcs may be inserted which are already in the graph. We could replace duplicated arcs by a single arc whose weight is the smaller of the two weights. However, this creates the problem of detecting arc duplications quickly in adjacency lists. We could also leave multiple arcs in the adjacency lists. The elimination algorithm is insensitive to multiple arcs, except for the storage and processing time they require: They cause no harm in the adjacency lists, and the algorithm will process two arcs with equal endpoints without notice. However, the analysis of the algorithm, which was given above, makes the assumption that from a vertex set of cardinality a to a vertex set of cardinality b there are at most ab arcs. Therefore, the inserted arcs would not be taken into account correctly, if this scheme were used.

Therefore, we have to switch to an adjacency matrix representation, at least for parts of the graph. Because of the space requirements, the matrix representation is certainly not suited for the original graph. Thus we must deal with two different representations in the same algorithm: an adjacency list for the graph as it is originally given, and a dynamical matrix representation for the inserted arcs:

The vertices on the path from the *root* to the current node in the tree are pushed on a stack, and they are associated with the rows and columns of the adjacency matrix in the order in which they appear on the stack. When the algorithm enters a new node, the corresponding number of rows and columns have to be added to the right and at the

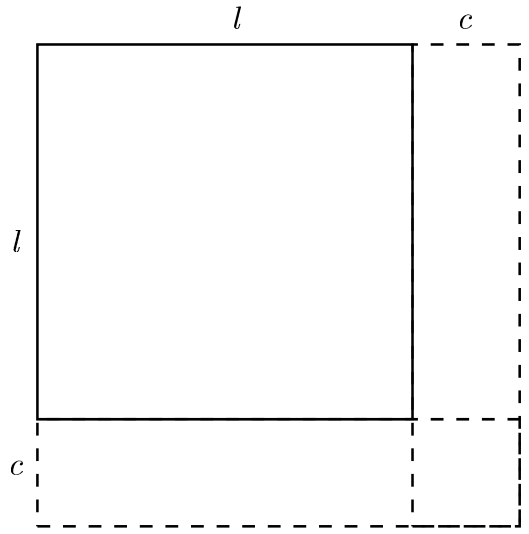

Figure 16. How the matrix grows and shrinks dynamically.

bottom of the matrix for the vertices stored in that node. (cf. Figure 16). The number of added elements $(2cl + c^2)$ corresponds exactly to the amount of fill-in that was accounted for in equation (7) of Section 2.1.3.4.2.

It was shown above in (8) that the total number l of vertices stored on the path from the root to any node is bounded by a falling geometric series starting with $\beta\sqrt{n}$. Hence the maximum dimension of the matrix is $O(\sqrt{n})$, and the storage requirement is $O(n)$.

When returning from a recursive procedure call, we pop the elements off the stack, and we simply shrink the dimensions of the matrix and forget what was outside. The time of $O(2cl + c^2)$ for clearing this area to zero when allocating it (or when freeing it) is clearly dominated by the time needed for the elimination $(O(c(c+l)^2))$.

When we delete arcs that were inserted earlier, we certainly do not want to destroy the arcs that were originally in the graph. This problem is solved at the same time by our twofold representation of the arcs.

2.2. Testing whether a graph has a necklace tour

Again, as in Section 2.1, all theorems about necklace tours hold equally for 2-factors. We will usually formulate them in this way, and we will specialize them to necklace tours only in the summary.

2.2.1. Characterization of graphs which have necklace tours

To state our characterization, we formulate the optimal 2-factor problem as an integer program. Let $G = (P, G)$ be a weighted graph with $P = \{p_1, p_2, \ldots, p_n\}$. We define a binary variable x_{ij} for each edge $\{p_i, p_j\}$ of the graph, where $x_{ij} = 1$, if $\{p_i, p_j\}$ belongs to the 2-factor, and $x_{ij} = 0$, if it does not. The 2-factor problem can now be formulated as follows:

$$
(\mathrm{F}) \quad \begin{cases}
\text{minimize} \quad \sum_{\{p_i,p_j\}\in G} x_{ij} d_{ij} & (\mathrm{F}.1)\\[2ex]
\text{subject to} \quad \sum_{\{p_i,p_j\}\in G} x_{ij} = 2, \quad \text{for } i = 1, 2, \ldots, n, \text{ and} & (\mathrm{F}.2)\\[2ex]
\qquad x_{ij} \in \{0, 1\}, \quad \text{for } \{p_i, p_j\} \in G & (\mathrm{F}.3)
\end{cases}
$$

The linear programming relaxation of this problem, called the *fractional 2-factor problem* (FF) is obtained by replacing the constraints (F.3) by

$$
0 \le x_{ij} \le 1, \qquad \text{for } \{p_i, p_j\} \in G. \qquad\qquad (\mathrm{FF}.3)
$$

Theorem 24. *A graph G has a necklace 2-factor F if and only if the fractional 2-factor problem defined by* (F.1), (F.2), *and* (FF.3) *has a unique optimal solution which is integral. F is then the 2-factor corresponding to this solution.*

Proof: (i) First we prove the "only if" part. Let $r_1, r_2, \ldots, r_n$ be radii that realize F, that is, they fulfill (R). Then it is clear that these radii can be made to fulfill

$$
r_i + r_j > d_{ij}, \text{ if } \{p_i, p_j\} \in F, \text{ and} \qquad (\mathrm{R}.1')
$$
$$
r_i + r_j < d_{ij}, \text{ if } \{p_i, p_j\} \notin F, \qquad (\mathrm{R}.2)
$$

for example by adding $\delta/3$ to all radii, where $\delta = min\,\{\, d_{ij} - r_i - r_j \mid \{p_i, p_j\} \notin F\,\} > 0$. Now assume that $r_1, r_2, \ldots, r_n$ fulfill (R.1′) and (R.2). If we replace the distances d_{ij} in the objective function (F.1) by $d'_{ij} := d_{ij} - r_i - r_j$, then we get

$$
\sum_{\{p_i,p_j\}\in G} x_{ij} d'_{ij} = \sum_{\{p_i,p_j\}\in G} x_{ij}(d_{ij} - r_i - r_j) = \sum_{\{p_i,p_j\}\in G} x_{ij} d_{ij} - \sum_{\{p_i,p_j\}\in G} x_{ij} r_i - \sum_{\{p_i,p_j\}\in G} x_{ij} r_j
$$
$$
= \sum_{\{p_i,p_j\}\in G} x_{ij} d_{ij} - \sum_{i=1}^{n} \sum_{\{p_i,p_j\}\in G} x_{ij} r_i = \sum_{\{p_i,p_j\}\in G} x_{ij} d_{ij} - \sum_{i=1}^{n} 2 r_i,
$$

by (F.2). Thus, the new objective function

$$
(\mathrm{F}.1') \qquad \text{minimize} \sum_{\{p_i,p_j\}\in G} x_{ij} d'_{ij}
$$

differs from the old one by an additive constant, and the optimal solutions are the same with respect to both objective functions. But we have

$$
d'_{ij} = d_{ij} - r_i - r_j < 0, \text{ for } \{p_i, p_j\} \in F, \text{ and}
$$
$$
d'_{ij} = d_{ij} - r_i - r_j > 0, \text{ for } \{p_i, p_j\} \notin F.
$$

Therefore, the unique optimal solution of (F.1′) subject to (FF.3) is the solution corresponding to F:

$$\begin{aligned} x_{ij} &= 1, \quad \text{for } d'_{ij} < 0 \quad (\{p_i, p_j\} \in F), \text{ and} \\ x_{ij} &= 0, \quad \text{for } d'_{ij} > 0 \quad (\{p_i, p_j\} \notin F), \end{aligned}$$

and this solution also fulfills (F.2) and is thus the unique optimal solution of (F.1) subject to (F.2), and (FF.3).

(ii) Now, we prove the "if" part of the theorem. Assume that the vector $\hat{x} = (\hat{x}_{ij})$ is the unique optimal solution of (F.1), (F.2), and (FF.3). The following rather technical consideration shows that the coefficients d_{ij} in the objective function (F.1) can be changed slightly without destroying the optimality of $\hat{x}$. We need this in order to obtain strict inequalities in (R.2). Since the set M of solutions of the system of linear equations (F.2) and linear inequalities (FF.3) is bounded by the constraints (FF.3), this set is a polytope, and therefore the minimum of any linear objective function over M is attained at a at a basic solution of the linear program, i. e., at a vertex of the polytope M. Hence, $\hat{x}$ is a vertex of M. Since M has only finitely many vertices, we can define δ to be the minimum difference in objective function value (F.1) between $\hat{x}$ and any other vertex. If we replace d_{ij} by

$$d''_{ij} = \begin{cases} d_{ij} - \delta/n^2, & \text{if } \hat{x}_{ij} = 0, \text{ and} \\ d_{ij}, & \text{if } \hat{x}_{ij} = 1, \end{cases}$$

we get a new objective function

$$\text{minimize} \sum_{\{p_i,p_j\} \in G} x_{ij} d''_{ij}. \qquad \text{(F.1}''\text{)}$$

Now, we have, for all $x \in M$

$$\begin{aligned} \sum_{\{p_i,p_j\} \in G} x_{ij} d_{ij} \geq \sum_{\{p_i,p_j\} \in G} x_{ij} d''_{ij} &= \sum_{\{p_i,p_j\} \in G} x_{ij} d_{ij} - \sum_{\hat{x}_{ij}=0} \frac{\delta}{n^2} \\ &\geq \sum_{\{p_i,p_j\} \in G} x_{ij} d_{ij} - \binom{n}{2} \frac{\delta}{n^2} > \sum_{\{p_i,p_j\} \in G} x_{ij} d_{ij} - \frac{\delta}{2}. \end{aligned}$$

Thus, (F.1″) differs from (F.1) by at most $\delta/2$, for all $x \in M$. This implies that $\hat{x}$ is still optimal with respect to (F.1″) among all vertices of M, and hence it is an optimal solution of the linear program (F.1″), (F.2), and (FF.3).

It remains to find radii r_i that realize the 2-factor F corresponding to $\hat{x}$. Let us consider the dual program of (F.1″), (F.2), and (FF.3):

$$\begin{aligned} &\text{maximize} \sum_{i=1}^{n} 2r_i - \sum_{\{p_i,p_j\} \in G} \delta_{ij} \\ &\text{subject to } -\delta_{ij} + r_i + r_j \leq d''_{ij}, \text{ for } \{p_i, p_j\} \in G, \\ &\qquad\qquad \text{with } \delta_{ij} \geq 0 \text{ and } r_i \text{ arbitrary .} \end{aligned}$$

Here, the r_i are the dual variables associated with the vertices, and the δ_{ij} are associated with the constraints $x_{ij} \leq 1$. Complementarity implies that for $\hat{x}$ to be optimal, there must exist values r_i and δ_{ij} such that the following complementary slackness conditions hold:

$$\hat{x}_{ij} > 0 \text{ implies } r_i + r_j - \delta_{ij} = d''_{ij}, \text{ and}$$
$$\hat{x}_{ij} < 1 \text{ implies } \delta_{ij} = 0.$$

This means

$$\text{(I.1)} \qquad \begin{cases} \hat{x}_{ij} = 1 & \text{implies } r_i + r_j \geq d''_{ij} = d_{ij} \text{ and} \\ \hat{x}_{ij} = 0 & \text{implies } r_i + r_j \leq d''_{ij} = d_{ij} - \frac{\delta}{n^2} < d_{ij}. \end{cases}$$

Thus, the radii r_i, which are just the dual variables of the linear program, fulfill (R.1) and (R.2). Positivity (R.3) can be achieved with the help of Theorem 9. The resulting radii realize the 2-factor F corresponding to $\hat{x}$. ∎

Corollary 25. *A necklace 2-factor of a graph is the unique optimal 2-factor.* ∎

Corollary 26. *A necklace tour of a graph is the unique optimal tour.* ∎

2.2.2. Finding necklace tours and necklace 2-factors

By Theorem 24, we need only solve the fractional 2-factor problem specified by (F.1), (F.2), and (FF.3) in order to find a necklace 2-factor. Extending the reduction given in Derigs and Metz [1986] and Balas [1981] (cf. Fulkerson, Hoffman, and McAndrew [1965]) from the fractional m-factor problem to the assignment problem, we can reduce the fractional 2-factor problem to the $n \times n$ capacitated transportation problem specified below. For consistency, we define $d_{ii} = \infty$, for $1 \leq i \leq n$.

$$\text{(C)} \qquad \begin{cases} \text{minimize} \displaystyle\sum_{i=1}^{n}\sum_{j=1}^{n} y_{ij} d_{ij} \\[2ex] \text{subject to} \quad \displaystyle\sum_{j=1}^{n} y_{ij} = 2, \quad \text{for } 1 \leq i \leq n, \\[2ex] \qquad\qquad\quad \displaystyle\sum_{i=1}^{n} y_{ij} = 2, \quad \text{for } 1 \leq j \leq n, \text{ and} \\[2ex] \qquad\qquad\quad 0 \leq y_{ij} \leq 1, \quad \text{for } 1 \leq i, j \leq n. \end{cases}$$

This is essentially the same 2-factor problem, but on a bipartite graph with twice as many vertices. The complementary snecklace conditions for this problem and its dual are as follows:

$$\text{(I.2)} \qquad \begin{cases} y_{ij} > 0 \text{ implies } u_i + v_j \leq d_{ij}, \text{ and} \\ y_{ij} < 1 \text{ implies } u_i + v_j \geq d_{ij}. \end{cases}$$

If a set of values x_{ij}, r_i fulfills (F.2), (FF.3), and (I.1), a pair of primal and dual solutions y_{ij} and u_i, v_j fulfilling (C) and (I.2) can be defined by setting

$$y_{ij} := y_{ji} := x_{ij} \quad \text{and} \quad u_i := v_i := r_i.$$

Conversely, from a solution fulfilling (C) and (I.2), we can compute a solution of (F.2), (FF.3), and (I.1) as follows:

$$x_{ij} := \frac{y_{ij} + y_{ji}}{2} \quad \text{and} \quad r_i := \frac{u_i + v_i}{2}.$$

Since the values y_{ij} of an optimal basic solution of (C) are integral, it is necessary and sufficient for x_{ij} to be integral that $y_{ij} = y_{ji}$ for all i and j, i. e., that the solution is symmetric.

By a straightforward extension of Theorem 24, we get now the following result:

Theorem 27. *A necklace 2-factor exists if and only if the solution of the capacitated transportation problem (C) is symmetric and unique.* ■

Since the necklace 2-factor can be restricted to be contained in the subgraph $G^{(2)}$, we need only consider the arcs corresponding to the edges of $G^{(2)}$. (There are two arcs (p_i, p_j) and (p_j, p_i) corresponding to each edge $\{p_i, p_j\}$ of the original subgraph.)

For solving the transportation problem we can use the shortest augmenting path method (cf. Edmonds and Karp [1972], Tomizawa [1972], or Derigs [1988]). Since the total amount of flow is $2n$, we need at most $2n$ flow augmentations. Each flow augmentation involves a single-source shortest path computation. Since the graph is sparse, we use for the shortest path computations the variation of Dijkstra's algorithm with priority queues implemented as heaps. The time-complexity of this method is $O((e+n)\log n)$, where e is the number of edges of the graph. Since e is $O(n)$ in our case, this implies an $\mathrm{O}(n^2 \log n)$ time algorithm for finding necklace 2-factors in the plane. Note that this time matched by an algorithm of Gabow [1983] for (always) finding an optimal 2-factor.

Finally, we summarize the complexity results of this chapter obtained for finding necklace 2-factors and necklace tours.

2.3. Summary of results about the necklace condition

Theorem 28. *Given a 2-factor or a tour for n points in the plane, we can decide in $O(n^{3/2})$ time and $O(n)$ space whether it is a necklace 2-factor or a necklace tour. If it is a necklace 2-factor or a necklace tour, then positive radii that realize it can be constructed within the same time and space bounds.* ■

Theorem 29. *Given a set of n points in the plane, the necklace condition can be tested in $O(n^2 \log n)$ time and $O(n)$ space. If the necklace condition is satisfied, the necklace tour or necklace 2-factor and a set of radii that realize it can be constructed within the same asymptotic time and space bounds.* ■

Conclusion

We have identified two special classes of the Traveling Salesman Problem which are solvable in polynomial time. Although polynomial time is generally associated with "good" algorithms, we view our algorithms mainly as a contribution to Theoretical Computer Science. In the case of the N-line Traveling Salesman Problem, the degree of the polynomial seems to be too high for practical applications; and in the case of the necklace condition, the class of problem instances for which it holds is probably too restricted. The necklace condition might perhaps be used as a guide for the development of heuristics for finding suboptimal solutions of the Traveling Salesman Problem. (This would be the other of the two approaches to NP-hard problems mentioned in the introduction.)

An interesting open problem would be to extend the results about the necklace condition to higher dimensions. The results about the sparsity of the graphs $G^{(m)}$ (Section 2.1.2.1) carry over in a straightforward way. It is however not so easy to prove for example an $n^{2/3}$-separator theorem for $G^{(2)}$ of point sets in space, since the planar separator theorem can no longer be applied. This would be the first case of a general separator theorem for point sets in three dimensions (apart from regular grid graphs).

Acknowledgment

I thank Vladimir Deĭneko from the University of Dnepropetrovsk for drawing my attention to the necklace condition and to the results of Cutler for the N-line Traveling Salesman Problem during his stay at our institute in the academic year 1984/85.

References

Balas, E. [1981]
Integer and fractional matchings, *Ann. Discrete Math.* **11**, 1–13.

Bateman, P., and Erdős, P. [1951]
Geometrical extrema suggested by a lemma of Besicovitch, *Amer. Math. Monthly* **58**, 306–314.

Bentley, J. L., and Ottmann, T. A. [1979]
Algorithms for reporting and counting geometric intersections, *IEEE Trans. Comput.* **C-28**, 643–647.

de Bruijn, N. G. [1955]
Solution to exercises 17 and 18 (in Dutch), in: *Wiskundige opgaven met de oplossingen* **20**, pp. 19–20.

Carré, B. A. [1971]
An algebra for network routing problems, *J. Inst. Math. Appl.* **7**, 273–294.

Carré, B. A. [1979]
Graphs and networks, Clarendon, Oxford.

Chowla, S., Herstein, I. N., and Moore, W. K. [1951]
On recursions connected with symmetric groups I, *Canad. J. Math.* **3**, 328–334.

Cornuéjols, G., Fonlupt, J., and Naddef, D. [1985]
The Traveling Salesman Problem on a graph and some related integer polyhedra, *Mathematical Programming* **33**, 1–27.

Cutler, M. [1980]
Efficient special case algorithms for the N-line planar Traveling Salesman Problem, *Networks* **10**, 183–195.

Deĭneko, V. G., van Dal, R. and Rote, G. [1994]
The convex-hull-and-line traveling salesman problem: A solvable case, *Information Processing Letters* **51**, 141–148.

Deĭneko, V. G., and Woeginger, G. [1996]
The convex-hull-and-k-lines traveling salesman problem, *Information Processing Letters* **59**, 259–301.

Derigs, U. [1988]
Programming in networks and graphs — on the combinatorial background and near-equivalence of network flow and matching algorithms, Springer Verlag, Berlin, Lecture Notes in Economics and Mathematical Systems **300**.

Derigs, U., and Metz, A. [1986]
On the use of optimal fractional matchings for solving the (integer) matching problem, *Computing* **36**, 263–270.

Duffin, R. T. [1974]
On Fourier's analysis of linear inequality systems, *Math. Programming Stud.* **1**, 71–95.

Edelsbrunner, H., Guibas, L. G., and Stolfi, J. [1986]
Optimal point location in a monotone subdivision, *SIAM J. Comput.* **15**, 317–340.

Edelsbrunner, H., Rote, G., and Welzl, E. [1987]
Testing the necklace condition for shortest tours and optimal factors in the plane, in: *Automata, Languages, and Programming (Proc. 14th Int. Coll. on Automata, Languages, and Programming (ICALP), Karlsruhe, July 1987)*, ed. T. Ottmann, Springer Verlag, Berlin, Lecture Notes in Computer Science **266**, pp. 364–375.

Edmonds, J., and Karp, R. M. [1972]
Theoretical improvements in algorithmic efficiency for network flow problems, *J. Assoc. Comput. Mach.* **19** (1972), 248–264.

Eisenstat, S. C., Schultz, M. H., and Sherman, A. H. [1976]
Applications of an element model for Gaußian elimination, in: *Sparse Matrix Computations*, ed. J. R. Bunch and D. J. Rose, Academic Press, New York, pp. 85–96.

Fourier, J. [1827]
Mémoires de l'Académie des Sciences **7** (1827), 47–60.

Fulkerson, D. R., Hoffman, A. J., and McAndrew, M. M. [1965]
Some properties of graphs with multiple edges, *Canad. J. Math.* **17**, 166–177.

Gabow, H. N. [1983]
An efficient reduction technique for degree-constrained subgraph and bidirected network flow problems, in: *Proc. 15th Ann. ACM Symp. Theory Computing 1983*, pp. 448–456.

Garey, M. R., and Johnson, D. S. [1979]
Computers and Intractability — A Guide to the Theory of NP-Completeness, Freeman, San Francisco, 1979.

George, J. A. [1973]
Nested dissection of a regular finite element mesh, *SIAM J. Numer. Anal.* **10**, 345–367.

Gilmore, P. C., Lawler, E. L., and Shmoys, D. B. [1985]
Well-solved special cases, Chapter 4 of the book *The Traveling Salesman Problem*, ed. E. L. Lawler et al. [1985], pp. 87–143.

Gondran, M., and Minoux, M. [1979]
Graphes et algorithmes, Editions Eyrolles, Paris 1979. English translation: *Graphs and algorithms*, Wiley-Interscience, Chichester 1984.

Itai, A., Papadimitriou, C. H., and Szwarcfiter, J. [1982]
Hamiltonian paths in grid graphs, *SIAM J. Computing* **11**, 676–686.

Johnson, D. S., and Papadimitriou, C. H. [1985]
Computational complexity, Chapter 3 of the book *The Traveling Salesman Problem*, ed. E. L. Lawler et al. [1985], pp. 37–85.

Knuth, D. E. [1973]
The Art of Computer Programming, vol. 3: Sorting and Searching, Addison-Wesley.

Lawler, E. L. [1976]
Combinatorial Optimization: Networks and Matroids, Holt, Rinehart, and Winston, New York, 1976.

Lawler, E. L., Lenstra, J. K., Rinnooy Kan, A. H. G., and Shmoys, D. B. (ed.) [1985]
The Traveling Salesman Problem, Wiley-Interscience, Chichester.

Lee, D. T. [1982]
On k-nearest neighbor Voronoi diagrams in the plane, *IEEE Trans. Comput.* **C-31**, 478–487.

Lehmann, D. J. [1977]
Algebraic structures for transitive closure, *Theoret. Comput. Sci.* **4**, 59–76.

Lin, S., and Kernighan, B. W. [1973]
An effective heuristic algorithm for the Traveling Salesman Problem, *Oper. Res.* **21**, 498–516.

Lipton, R. J., Rose, D. J., and Tarjan, R. E. [1979]
Generalized nested dissection, *SIAM J. Numer. Anal.* **16**, 346–358.

Lipton, R. J., and Tarjan, R. E. [1979]
A separator theorem for planar graphs, *SIAM J. Appl. Math.* **36**, 177–189.

Moser, L., and Wyman, M. [1955]
On solutions of $x^d = 1$ in symmetric groups, *Canad. J. Math.* **7**, 159–168.

Megiddo, N. [1983]
Towards a genuinely polynomial algorithm for linear programming, *SIAM J. Comput.* **12**, 347–353.

Papadimitriou, C. H. [1977]
The Euclidean TSP is NP-complete, *Theoret. Comput. Sci.* **4**, 237-244.

Preparata, F. P., and Shamos M. I. [1985]
Computational Geometry — An Introduction, Springer Verlag, New York, 1985.

Ratliff, H. D., and Rosenthal, A. S. [1983]
Order-picking in a rectangular warehouse: a solvable case of the Traveling Salesman Problem, *Operations Research* **31**, 507–521.

Reifenberg, E. R. [1948]
A problem on circles, *Math. Gaz.* **32** (1948), 290–292.

Rockafellar, R. T. [1984]
Network Flows and Monotropic Optimization, Wiley-Interscience, New York, 1984.

Rote, G. [1988]
The N-line Traveling Salesman Problem, Bericht 1988-109, Technische Universität Graz, Institut für Mathematik, January 1988.

Rothe, H. A. [1800]
Über Permutationen, in Beziehung auf die Stellen ihrer Elemente. Anwendung der daraus abgeleiteten Sätze auf das Eliminationsproblem, in: *Sammlung combinatorisch-analytischer Abhandlungen, 2. Sammlung*, ed. C. F. Hindenburg. Leipzig, bei Gerhart Fleischer dem Jüngeren, pp. 263–305.

Sanders, D. [1968]
On extreme circuits, Ph. D. thesis, City College of New York, 1968.

Shostak, R. [1981]
Deciding linear inequalities by computing loop residues, *J. Assoc. Comput. Mach.* **28** (1981), 769–779.

Supnick, F. [1970]
A class of combinatorial extrema, *Ann. New York Acad. Sci.* **175** (1970), 370–382.

Tomizawa, N. [1972]
On some techniques useful for solution of transportation network problems, *Networks* **1** (1972), 179–194.

Zimmermann, U. [1981]
Linear and combinatorial optimization in ordered algebraic structures, in: Ann. Discrete Math., vol. **10**, 380 pp.

Contents