Algorithmica (1995) 13: 357–380

Algorithmica

New Primal and Dual Matching Heuristics[1]

M. Jünger[2] and W. Pulleyblank[3]

Abstract. We describe a new heuristic for constructing a minimum-cost perfect matching designed for problems on complete graphs whose cost functions satisfy the triangle inequality (e.g., Euclidean problems). The running time for an n node problem is $O(n \log n)$ after a minimum-cost spanning tree is constructed. We also describe a procedure which, added to Kruskal's algorithm, produces a lower bound on the size of any perfect matching. This bound is based on a dual problem which has the following geometric interpretation for Euclidean problems: Pack nonoverlapping disks centered at the nodes and moats surrounding odd sets of nodes so as to maximize the sum of the disk radii and moat widths.

Key Words. Matching, Heuristics, Moat packing, Minimum spanning tree.

1. Introduction. In an undirected graph $G = (V, E)$ with edge weights w_e for $e \in E$ the *matching problem* consists of determining a set $M \subseteq E$ of pairwise nonadjacent edges with minimum (maximum) total weight $w(M) := \sum_{e \in M} w_e$. The *matching M* is called *perfect* if M meets every node in V, i.e., $V = \bigcup M$. Edmonds (1965b) has given an algorithm for the (perfect) matching problem which can be implemented to run in $O(|V|^3)$ time. For many practical applications on large graphs, the running time of Edmonds's algorithm is extremely long. So a considerable amount of work has been devoted to the study of faster approximative algorithms. In particular, many heuristics for the Euclidean perfect-matching problem in the plane have been proposed and analyzed with respect to their performance and complexity. Here, the nodes in V are assumed to be points in the Euclidean plane and the distance d_{ij} between two points $i = (x_i, y_i)$ and $j = (x_j, y_j)$ is assumed to be the Euclidean distance $d_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$. For simplicity, we assume that all points in V lie in the unit square, i.e., each point $i = (x_i, y_i)$ has $0 < x_i < 1$ and $0 < y_i < 1$. If $e = uv$ is an edge of a graph defined on these points, we let $d(e)$ denote d_{uv}.

Many real-world combinatorial optimization problems are Euclidean problems. A notable example is the minimization of pen movement on a mechanical plotter, see Reingold and Tarjan (1981) and Iri *et al.* (1983). Furthermore, many combinatorial optimization problems arising in the context of VLSI design are Euclidean.

[1] This research was supported in part by the Natural Sciences and Engineering Research Council of Canada.
[2] Institut für Informatik, Universität zu Köln, Pohligstrasse 1, D-5000 Köln 51, Germany.
[3] International Business Machine Corporation, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, NY 10598, USA.

Received February 3, 1992; revised May 20, 1993. Communicated by C. H. Papadimitriou.

A fast approximative Euclidean matching algorithm is also useful for efficient versions of Christofides' heuristic for the traveling salesman problem (see Christofides, 1976; Cornuéjols and Nemhauser, 1978) and for Chinese postman heuristics (see Grigoriadis and Kalantari, 1985a). In addition, such heuristics can be expected to produce good starting solutions in exact primal matching algorithms, like those by Cunningham and March (1978), Derigs (1986), and Grötschel and Holland (1985).

Approximative algorithms for the Euclidean matching problem on the unit square (as well as the more general case where the edge weights are only assumed to satisfy the triangle inequality) have been designed by many authors. The heuristics may be classified with respect to their worst-case time complexity, average-case time complexity, worst-case weight of the heuristic solution, and expected weight of the heuristic solution, assuming a certain (usually uniform) distribution of the n points on the unit square. A compilation of most known results for a large selection of matching heuristics can be found in Avis (1983).

It follows from the results of Beardwood *et al.* (1959) that a constant μ exists such that the expected weight of a Euclidean matching of n uniformly distributed points in the unit square is $\mu\sqrt{n}$ (see Papadimitriou, 1977). Papadimitriou also showed that $0.25 \leq \mu \leq 0.40106$ and conjectured $\mu \approx 0.35$. Based on extensive computational experiments, Weber and Liebling (1985) report the approximate value $\mu \approx 0.3189$.

The best expected performance we could find in the literature for heuristics is that of the STRIP heuristic (Papadimitriou, 1977) which runs in time $O(n \log n)$. More precisely, if P is a set of n points uniformly distributed in the unit square, then the expected weight of the matching produced by STRIP is $0.474\sqrt{n} + o(\sqrt{n})$ (see Supowit *et al.*, 1980). An upper bound on the worst-case performance of $0.707\sqrt{n} + O(1)$ is also derived in the same reference.

Computational results for $O(n)$ and $O(n \log n)$ heuristics can be found in Iri *et al.* (1983). Here it turns out that the STRIP heuristic outperforms all tested linear-time heuristics when applied to large real-world problems.

Another heuristic of time complexity $O(n \log n)$, called the Minimum Spanning Tree Heuristic (MSTH) has been proposed by Papadimitriou (see Supowit *et al.*, 1980). MSTH works for the more general case where the n points are only assumed to satisfy the triangle inequality (but then the time complexity is $O(n^2)$), and the relative performance guarantee $w_{\mathrm{MSTH}}/w_{\mathrm{OPT}} \leq n/2$ is proved. Here w_{OPT} and w_{MSTH} stand for the minimum and approximative weights, respectively. In addition, the bound $n/2$ can be achieved asymptotically.

Grigoriadis and Kalantari (1985a, b) observe that "MSTH is optimal in the sense that, for most computational models of interest, the worst-case time complexity of any heuristic that produces an approximative solution with a finite ratio bound is $\Omega(n^2)$ for general weights and $\Omega(n \log n)$ for Euclidean problems." Based on our own computational experiments, we found that Papadimitriou's heuristic delivers matchings with an average weight of about $0.358\sqrt{n}$ for n uniformly distributed points, i.e., significantly better values than STRIP, which is of the same time complexity.

In Section 2 we propose another $O(n \log n)$ heuristic for minimum Euclidean matching, which is also based on a minimum-length spanning tree. Our main motivation is the following. It delivers solutions with an observed average weight of about $0.338\sqrt{n}$ for n uniformly distributed points, yet runs in about the same time as MSTH for our test problems of sizes up to 10,000 nodes. If we assume that Weber's and Liebling's estimate for the weight of the minimum matching of 0.3189 is correct, then this would mean that we are only about 6% above the optimum. In Section 3 we describe a companion heuristic which produces a lower bound on the value of the optimum solution. This gives an "individualized" quality guarantee of the kind "The heuristic solution is at most p percent above the optimum." For uniformly distributed random problems we obtain $p \approx 22$. However, the quality guarantee is much better on many real-world problems.

2. A Primal Heuristic. Our method proceeds as follows. If the number of points is less than some threshold, usually four, six, or eight, then the problem is solved exactly. If not, a minimum-cost spanning tree T is constructed on the points. The longest nonpendent edge uv of T is found and removed, thereby partitioning T into T_u and T_v, where $u \in V(T_u)$ and $v \in V(T_v)$. If the number of nodes in each of T_u and T_v is even, then the algorithm is applied recursively to the nodesets of each of T_u and T_v. The result is the union of the two perfect matchings thereby found.

If the number of nodes in each of T_u, T_v is odd, then we proceed as follows (see Figure 2.1). First, apply the algorithm to $V(T_u) \cup \{v\}$. In the matching M_1 produced, some node $w \in V(T_u)$ will be matched with v. Now recursively apply the algorithm to $V(T_v) \cup \{w\}$. Let M_2 be the matching produced. We return the matching $(M_1 \setminus \{vw\}) \cup M_2$.

Note that we do not have to recompute the minimum-cost spanning tree for each recursive call. If $|V(T_u)|$ and $|V(T_v)|$ are even, then T_u and T_v will be minimum spanning trees of $V(T_u)$ and $V(T_v)$, respectively. If $|V(T_u)|$ and $|V(T_v)|$ are odd, then T_u plus the edge uv will be a minimum-cost spanning tree of $V(T_u) \cup \{v\}$. In order to obtain a minimum-cost spanning tree of $T_v \cup \{w\}$, we must find the node t of T_v nearest to w. We then add edge tw to T_v. We refer to this algorithm as DUST (Decomposition Using Spanning Trees).

Our main theorem of this section is that DUST can be implemented so that its running time for Euclidean perfect-matching problems is $O(n \log n)$. More generally, its running time for any perfect-matching problem is $O(n \log n)$, after a minimum-cost spanning tree has been found in the original graph. We first describe the necessary data structures.

Let V be a finite set of n points in $\mathbf{R}^2$. The *Voronoi diagram* is a set of $|V|$ regions, where the region S_v for $v \in V$ consists of all those points in $\mathbf{R}^2$, for which v is the nearest member of V. Then each S_v will be a (closed) convex polyhedron in $\mathbf{R}^2$. Two regions S_v, S_w for which $|S_v \cap S_w| > 1$ are called *adjacent*. The *Delaunay triangulation* is the graph constructed on the node set of V where two members of V are adjacent if and only if the corresponding regions are adjacent. Note that the "Delaunay triangulation" is planar, but not necessarily a triangulation.

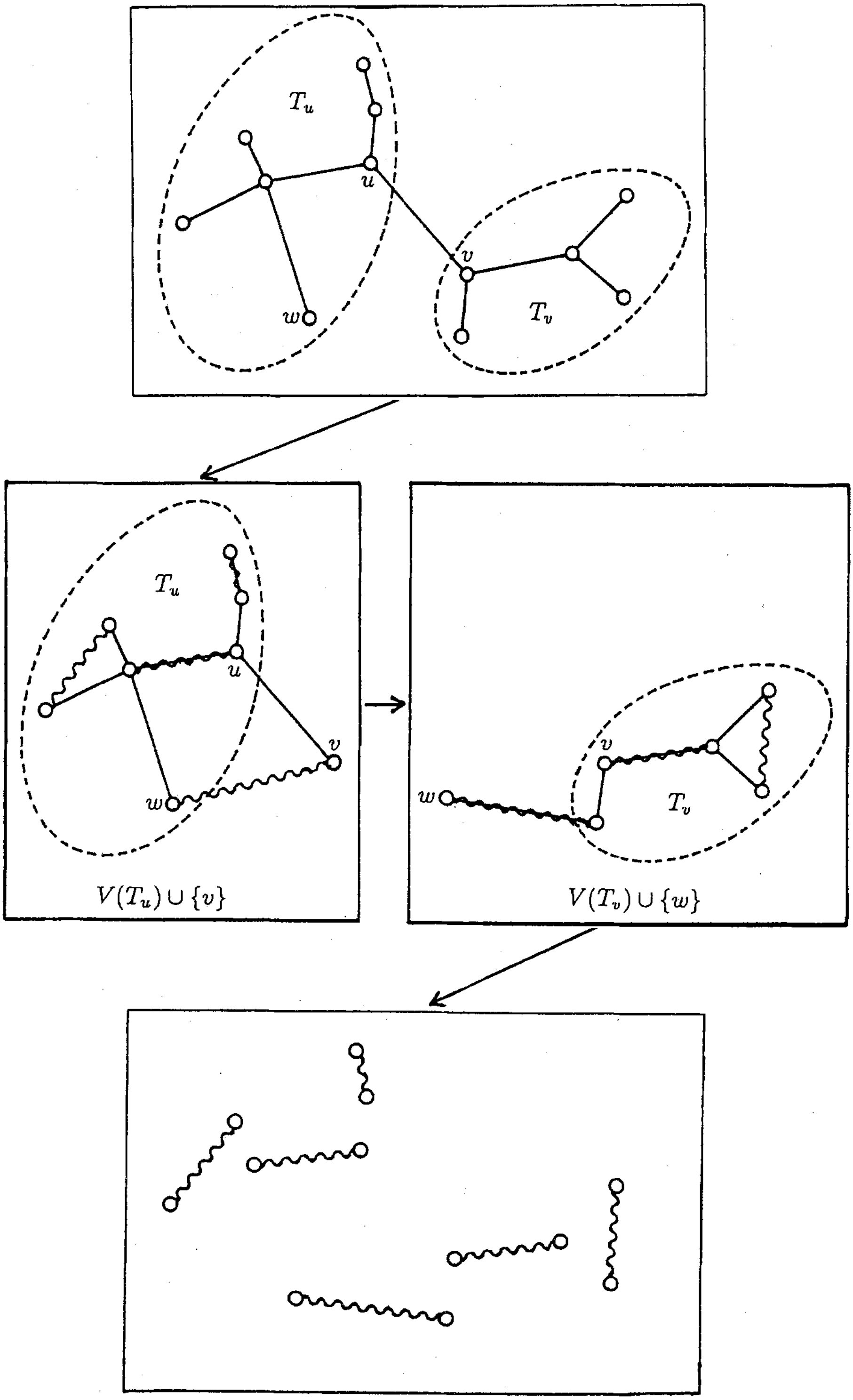

Fig. 2.1. Odd split.

Shamos (1975) showed that the Voronoi diagram (and, therefore, the Delaunay triangulation) can be constructed in time $O(n \log n)$, where all arithmetic calculations are assumed to take constant time. Moreover, the edgeset of a minimum-cost spanning tree of V is contained in the edgeset of the Delaunay triangulation. (See also Mehlhorn, 1984.) Ohya *et al.* (1984) describe an algorithm for constructing the Voronoi diagram whose running time is $O(n^2)$ in the worst case, but whose experimentally observed running time is only $O(n)$ on the average when applied to n uniformly distributed points in the Euclidean unit square. This was the method we used in our tests. Recently Sugihara and Iri (1988), Sugihara (1988), and Jünger *et al.* (1991) have developed variations of this method which avoid the problems of numerical instability encountered in the original.

Tarjan (1983) describes how Kruskal's algorithm for obtaining a minimum-cost spanning tree in a graph $G = (V, E)$ can be implemented in time $O(|E| \log |E|)$. Initially, each node is considered to be a separate component. The edges are considered in order of increasing cost. If an edge joins two nodes of the same component, then it is discarded. If it joins two nodes of different components, then the edge is added to the tree and the components are merged.

Before giving the detailed description of the algorithm, we note two preliminary facts.

LEMMA 2.1. *If V is a set of eight or more points in the Euclidean plane, then any Euclidean minimum-length spanning tree contains a nonpendent edge.*

PROOF. Let T be a minimum-cost spanning tree and suppose there is no nonpendent edge. Then T is a star with a center node r, and since $|V \backslash \{r\}| \geq 7$, there are nodes w and v such that the edges in T joining w and v to r form an angle α of less than 60°. See Figure 2.2. Let $x = d_{vr}$ and let $y = d_{wr}$ and assume $x \leq y$. Then $d_{vw}^2 = (x \sin \alpha)^2 + (y - x \cos \alpha)^2 = x^2 + y^2 - 2xy \cos \alpha$. Since $0° \leq \alpha < 60°$, $\cos \alpha > \frac{1}{2}$. Using this, plus the fact that $y \geq x$, we obtain $d_{vw}^2 < y^2$, which contradicts T being a minimum-cost spanning tree. □

The second lemma shows that when we perform an odd split in the course of the algorithm, we obtain a minimum-cost spanning tree by adding the one extra node to its nearest neighbor on the other side.

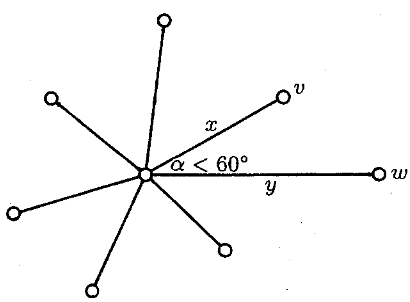

Fig. 2.2. Necessity of nonpendent edge.

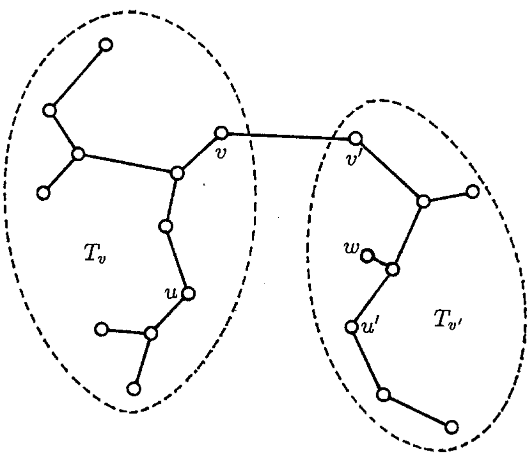

Fig. 2.3. Proof of Lemma 2.2.

LEMMA 2.2. *Let T be a minimum-cost spanning tree and let vv' be an edge. Let $u \in V(T_v)$ and let u' be the nearest node in $V(T_{v'})$. Then $\bar{T} = T_{v'} \cup \{uu'\}$ is a minimum-cost spanning tree on $V(T_{v'}) \cup \{u\}$.*

PROOF. Suppose not. Then there is an edge st which when added to $\bar{T}$ creates a cycle C containing an edge longer than st. Because T was a minimum-cost spanning tree, $T_{v'}$ is a minimum-cost spanning tree on $V(T_{v'})$, so C must contain uu', plus an edge uw between u and some node of $T_{v'}$ different from u'. (That is, $st = uw$.) Let pq be the longest edge in C. Then $d_{pq} > d_{uw} \geq d_{uu'}$. The edge pq must either belong to the path in $T_{v'}$ joining w to v' or the path joining u' to v', since it is contained in the path joining w to u'. However, this means that it belongs to the cycle created when we add either uw or uu' to T, see Figure 2.3. This contradicts T being a minimum-cost spanning tree. □

In order to achieve the claimed running time, we construct the following structure while the minimum-cost spanning tree algorithm is being run. Let $E = \{e_1, e_2, \ldots, e_{n-1}\}$ be the edgeset of the minimum-cost spanning tree, where $d(c_i) \leq d(c_{i+1})$ for $1 \leq i \leq n-2$. A *leaf* of T is an edge of T for which an endnode has degree one in T. Let V be the nodeset of T, i.e., of our original problem.

We use an auxiliary rooted binary tree $B(T)$ to locate the longest (nonpendent) edge of T. We refer to the nodes of $B(T)$ as *b-nodes*. We construct $B(T)$, during the running of Kruskal's algorithm as follows:

(2.1) If a component K of the forest constructed so far has no edges, then the corresponding tree $B(K)$ is empty.

(2.2) If we merge two components by adding an edge e_j, then we construct a new b-node corresponding to e_j, and give it children corresponding to the binary trees representing the two merged components.

Then $B(T)$ will satisfy the following:

(2.3) The b-nodes of $B(T)$ correspond to the edges of T.

(2.4) For each b-node j of $B(T)$, if $e(j)$ is the corresponding edge of T, then $d(e(j)) \geq d(e(i))$, for any edge $e(i)$ corresponding to any descendent b-node i of j.

Note that (2.4) implies that the longest edge of T occurs at the root of $B(T)$. See Figure 2.4. The letter beside each node is for identification and the number beside each edge is its length.

The structure $B(T)$ is only used to locate the longest edge of T. We maintain a separate "graph representation" of T which allows us to manipulate its structure. It must permit us to determine the neighbors of any node, or, equivalently, the incident edges, plus add and delete nodes and edges efficiently.

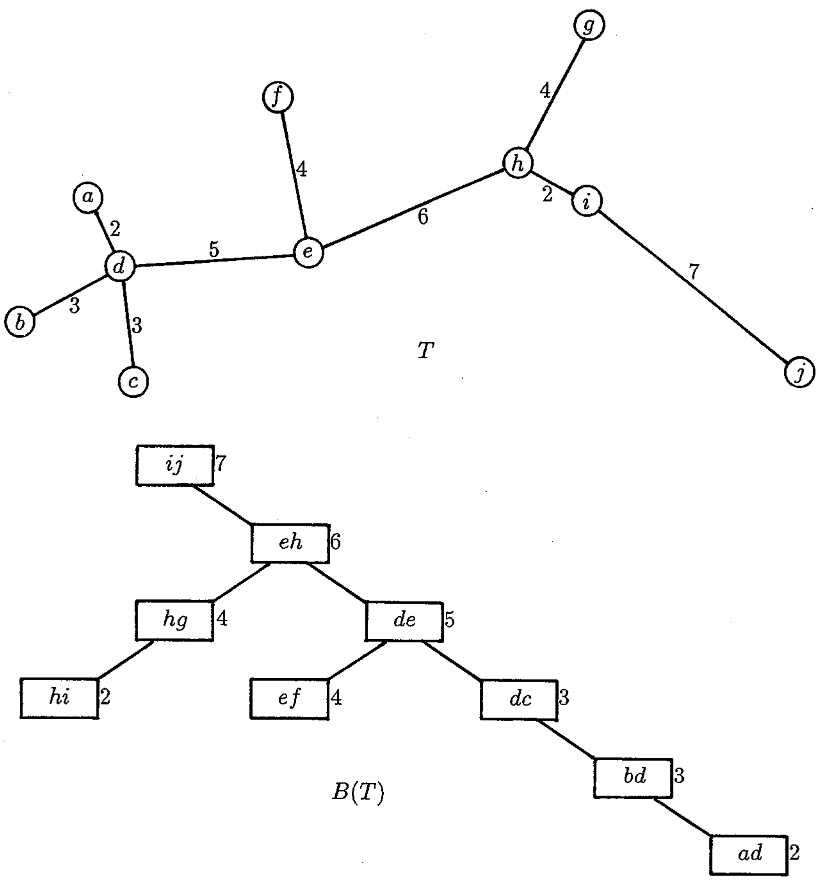

Fig. 2.4. Auxiliary tree $B(T)$.

When a node w is added to a subtree T_v during an odd split, we mark w, to indicate this fact. This is because we know that the length of the unique edge of the tree incident with w is longer than any nonpendent edge of the tree, and we will have to recognize this fact at one point.

The main procedure of our algorithm is called DUST (Decomposition Using Spanning Trees). This makes use of a recursive subroutine MATCH(T, $B(T)$). The subroutine is passed the spanning tree T as well as its associated structure $B(T)$. It will return the set of pairs of nodes comprising a perfect matching on V. We may assume that this matching is represented by a function PAIR(v) which gives, for each node v, the node with which it is matched.

DUST (Decomposition Using Spanning Trees)
Input: An even cardinality set S of points in the Euclidean plane.
Output: A perfect matching of S.

1. Construct the Delaunay triangulation for S.
2. Construct a minimum-cost spanning tree T on S, and the associated binary tree $B(T)$.
3. Call MATCH(T, $B(T)$).

Procedure MATCH(T, $B(T)$)

1. If $|V| \leq$ limit, for some fixed limit (4, 6, or 8 usually) we use an exact algorithm (e.g., enumeration) to construct a minimum-cost perfect matching and return.
2. ($|V| >$ limit.) Let $l(j) = uv$ be the edge corresponding to the root j of $B(T)$. Let T_u and T_v be the subtrees of T, containing u and v, respectively, obtained when we delete edge uv from T. We now use the graph representation of T to scan these two subtrees in parallel. As soon as we know the number k of nodes in the smaller part, we terminate the scan. If $k = 1$, i.e., uv is a leaf of T, we replace j with its unique child in $B(T)$ and repeat the scan. If $k \geq 2$ is even, we go to Step 3. If $k \geq 3$ is odd, we go to Step 4.
3. (Even Split.) Let B^u and B^v be the child subtrees in $B(T)$ of j. (Either or both may be empty, but if either T_u or T_v has at least eight nodes, then the corresponding B^u or B^v will be nonempty (see Lemma 2.1).) Call MATCH(T_u, B^u) and MATCH(T_v, B^v) and return the union of the two matchings produced.
4. (Odd Split.) Assume $|V(T_u)| \geq |V(T_v)|$. Let B^u and B^v be the child subtrees in $B(T)$ of j. Let T'_u be obtained from T_u by adjoining the edge uv. Call MATCH(T'_u, B^u). In the matching M_1 returned, let $w =$ PAIR(v).

   Scan T_v to find the node w' nearest to w. Let T'_v be formed by adding ww' to T_v. Mark w to indicate that it was added during an odd split. By Lemma 2.2, T'_v is a minimum-cost spanning tree on its nodeset.

   Check to see whether w' is a pendent node of T_v added at a previous iteration of Step 4. If so, we know that the length of the edge $w't$ joining w' to another node of T_v is greater than any nonpendent edge, and so, after attaching ww', this edge $w't$ will be the next chosen for a split, which will be even. Add the edge ww' to the matching, and remove $w't$ from T_v. Then call MATCH($T_v \backslash \{w't\}, B_v$). Let M_2 be the matching returned. Return $M_1 \backslash \{vw\} \cup \{ww'\} \cup M_2$.

If w' is not a pendent node of T_v, then we simply call

$$\text{MATCH}(T_v \cup \{ww', B_v\}).$$

Let M_2 be the matching returned. Return $M_1 \backslash \{v, w\} \cup M_2$.

Before proving an upper bound on the running time of this algorithm, we make one preliminary observation regarding the auxiliary data structure $B(T)$. It is never changed in the course of the algorithm. Indeed, in the entire course of the algorithm, it is scanned once. The important property is that the highest node with two children always corresponds to the largest nonpendent edge in the appropriate tree. If we add a new node to a tree during an odd split, then the new node becomes pendent, so we do not have to update $B(T)$. However, if we add a new node w during an odd split, it may happen that the node w' to which it was attached had been added during a previous execution of Step 4, and so, when we add w, w' is no longer a leaf and this fact should be reflected in $B(T)$. This is why, in this case, we effectively perform the next recursive call immediately. The next split would be the even split on the edge $w't$. Therefore one recursive call would be on $\{w, w'\}$, which would be matched together. We simply pair them off, then make the next call on the tree with these nodes removed. In this way, $B(T)$ still performs properly.

THEOREM 2.3. *When DUST is applied to a set V of n (even) nodes, the running time is $O(n \log n)$.*

PROOF. Let $t(n)$ be the time required by MATCH when applied to a set of n nodes, except for the time scanning $B(T)$. Then $t(n)$ is defined by the recurrence

$$t(n) \leq \begin{cases} c & \text{for} \quad n \leq 8, \\ t(k+1) + t(n-k+1) + \text{extra} & \text{for} \quad n > 8. \end{cases}$$

Here "extra" denotes the time incurred apart from the recursive calls. We assume that the split results in a tree T_u with $n-k$ nodes and a tree T_v with k nodes, and $k \leq n-k$.

The time accounted for in "extra" includes the following:

(i) Location of the largest edge in T.
This is accounted for separately.

(ii) Determination of which of T_u and T_v is smaller.
This is done by scanning both trees in parallel, and stopping when one is completed. Thus this time is $O(k)$.

(iii) Choice of the node w' to which $w = \text{PAIR}(v)$ must be attached.
This is done by scanning the smaller part, so the time is $O(k)$.

(iv) Updating T_u and T_v by adding or removing at most two edges.
This requires constant time.

Therefore "extra" is $O(k)$, so $t(n)$ satisfies the recurrence

$$(2.5)\qquad \begin{aligned} &t(n) \le c' \quad \text{for} \quad n \le 8,\\ &t(n) \le t(k+1) + t(n-k+1) + c'k \quad \text{for} \quad n > 8, \end{aligned}$$

for some constant c'. We prove inductively that this implies

$$(2.6)\qquad t(n) \le c'(n-2)\log(n-2) + c' \quad \text{for all} \quad n \ge 8,$$

proving the result.

Substitute (2.6) for k and $n-k$ into (2.5). We obtain

$$t(n) \le c'(k-1)\log(k-1) + c'(n-k-1)\log(n-k-1) + c'k.$$

Since $k \le n/2$,

$$\log(k-1) \le \log((n-2)/2) = \log(n-2) - 1.$$

Since $k \ge 1$,

$$\log(n-k-1) \le \log(n-2).$$

Therefore,

$$\begin{aligned} t(n) &\le c'(k-1)[\log(n-2) - 1] + c'(n-k-1)\log(n-2) + c'k\\ &= c'(n-2)\log(n-2) + c', \end{aligned}$$

as required.

The other time required by DUST includes constructing the Delaunay triangulation, finding the minimum-cost spanning tree, and scanning $B(T)$. The first two activities require $O(n \log n)$ and the latter $O(n)$. Therefore, the total running time is $O(n \log n)$ as asserted. □

While developing this algorithm, we tried several variations in order to simplify the odd split. For example, when we have an odd split uv into T_u and T_v, we could simply apply MATCH to $T_u \cup \{uv\}$ and $T_v \cup \{uv\}$. Let M_1 and M_2 be the two matchings produced. If $uv \in M_1$ and M_2, then return $M_1 \cup M_2$. If $uv \in M_1$ but $uv \notin M_2$, then return $M_1 \backslash \{uv\} \cup M_2$. If uv belongs to neither M_1 nor M_2, then remove the edges incident with u and v from M_1 and M_2, yielding M_1' and M_2'. Now return $M_1' \cup M_2'$ plus a minimum-cost perfect matching on u, v together with the other four unmatched nodes.

This variation has the advantage that the extra work required during an odd split now has constant time. Therefore, the running time of the algorithm, after construction of the Delaunay triangulation and minimum-cost spanning tree, becomes linear in n. However, our computational experience was that this variation did not provide as good solutions as DUST.

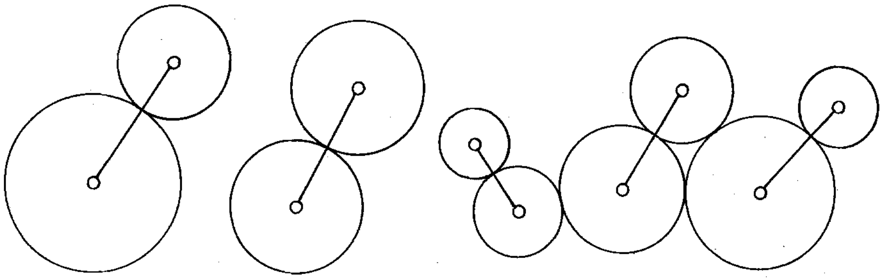

Fig. 3.1. Optimal disk packing and perfect matching.

3. Individual Quality Guarantees. In this section we introduce a dual geometric problem to that of constructing a minimum-length perfect matching. It has two important properties. First, every feasible solution provides a lower bound on the length of any perfect matching. Second, if we find a best feasible solution, then this bound will exactly equal the minimum length of a perfect matching.

In general, we do not know a better way to solve this dual problem optimally than to solve the minimum perfect-matching problem exactly. However, in the second part of this section we describe a method for constructing a feasible solution to this dual problem based upon Kruskal's minimum-cost spanning-tree algorithm. Our experience has been that these bounds are usually quite good for Euclidean problems. The gap between the length of the matching produced by DUST and the value of this bound is typically 10%–20%. The computational results are discussed in the last section.

Consider a Euclidean minimum-length perfect-matching problem for an even cardinality set V of points. Suppose, for each $v \in V$, we construct a closed disk D_v centered at v of radius r_v, such that no two disks overlap, that is, have interior points in common. If we represented a perfect matching M of V by line segments between the matched pairs of nodes, then each disk D_v would contain a length of at least r_v of the segment incident with v. Therefore the length of this matching is at least $\sum_{v \in V} r_v$. Thus a first dual problem would be to construct nonoverlapping disks centered at the points, such that the sum of the radii is maximized.

Consider the examples of Figures 3.1 and 3.2. In the first case we were able to construct an optimal disk packing which proved optimality of the displayed perfect matching. In the second, although we exhibit an optimal packing, the sum of the

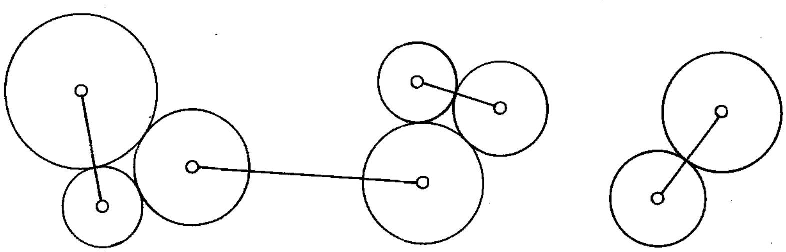

Fig. 3.2. Optimal disk packing bound not tight.

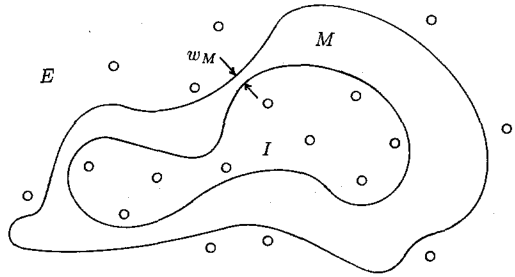

Fig. 3.3.

radii is much less than the minimum length of a perfect matching. (Note that for the bound to be tight, each matching line segment must be completely accounted for by disk radii.)

In order to close this gap, we introduce one new type of object. Let I and $\hat{M}$ be compact subsets of the plane such that I is contained in the interior of $\hat{M}$. Let $E = \mathbf{R}^2\backslash\text{interior}(\hat{M})$. Let $M = \hat{M}\backslash\text{interior}(I)$. We say that M is a *moat* of *width* w_M with *interior* I and *exterior* E if

(3.1) each of I, E contains an odd number of members of V and M contains no members of V,

(3.2) The infimum of the distances between points $x \in I$ and $y \in E$ is w_M,

see Figure 3.3. Our definition permits moats to be very general. However, the ones we use have a special structure which we describe later.

Since each of I, E contains an odd number of points of V, in any perfect matching, at least one member of V in I must be matched with a member in E. Hence, some segment representing the matching must have length at least w_M in M. Therefore, if $\mathscr{D}$ is a set of disks centered at the members of V and $\mathscr{M}$ is a set of moats such that no two members of $\mathscr{D} \cup \mathscr{M}$ overlap each other, then $\sum_{v\in V} r_v + \sum_{M\in\mathscr{M}} w_M$ is a lower bound on the length of a minimum-length perfect matching. We shall see that this bound is tight for an optimum disk/moat packing.

Edmonds (1965a, b) proved that the minimum-length perfect-matching problem can be solved as the linear program

$$\min \sum_{u,v\in V} d_{uv} x_{uv}$$

$$\sum_{u\in S,\, v\in V\backslash S} x_{uv} \geq 1 \qquad \text{for all} \quad S \in \mathscr{Q},$$

$$\sum_{v\in V,\, u\neq v} x_{uv} = 1 \qquad \text{for all} \quad u \in V,$$

$$x_{uv} \geq 0 \qquad \text{for all} \quad u, v \in V, \quad u \neq v,$$

where $\mathscr{Q} = \{S \subseteq V \mid 3 \leq |S| \leq |V| - 3, |S| \text{ odd}\}$. Edmonds proved that every basic solution to this linear program is 0–1 valued, i.e., the incidence vector of a perfect matching.

The dual linear program is

$$(3.3)\qquad \begin{aligned} &\max \sum_{v\in V} r_v + \sum_{S\in\mathcal{Q}} w_S \\ &r_u + r_v + \sum_{S\in\mathcal{Q},\,|\{u,v\}\cap S|=1} w_S \le d_{uv} \qquad \text{for all} \quad u, v \in V, \\ &\qquad\qquad w_S \ge 0 \qquad \text{for all} \quad S \in \mathcal{Q}. \end{aligned}$$

Edmonds's proof consisted of a polynomial algorithm which solved both programs. The optimum solution r^*, w^* produced by the dual problem has the following important property. The subset of those $S \in \mathcal{Q}$ for which $w_S^* > 0$ forms a nested family. That is, if $w_S^* > 0$ and $w_T^* > 0$, then $S \cap T = \varnothing$ or $S \subseteq T$ or $T \subseteq S$.

THEOREM 3.1. *Let M be a minimum-length perfect matching for an even cardinality set V of points in the Euclidean plane. Then a set $\mathcal{D}$ of disks centered at the members of V and $\mathcal{M}$ of moats exists such that the members of $\mathcal{D} \cup \mathcal{M}$ are nonoverlapping and $\sum_{D_v\in\mathcal{D}} r_v + \sum_{S\in\mathcal{M}} w_S$ equals the length of M.*

PROOF. Apply Edmonds's theorem to obtain a minimum-length perfect matching M^* and an optimum dual solution r^*, w^* such that w^* is nested. First, suppose that $r_v^* < 0$ for some $v \in V$. We could permit disks of negative radius, but this would detract from the geometric result, and is unnecessary. Instead we modify the solution so that all radii become nonnegative.

Let $\mathcal{S} = \{S \in \mathcal{Q} \mid w_S^* > 0\}$. Define a new nested family $\mathcal{S}'$ equal to $\{\{u\} \mid u \in V\backslash\{v\}\}$, plus, for each $S \in \mathcal{S}$, whichever of S or $V\backslash S$ does not contain v. For convenience of notation, for $u \in V\backslash\{v\}$, define $w_{\{u\}}^* = r_u^*$. Now let $U = \{u \in V\backslash\{v\} \mid \sum_{S\in\mathcal{S}',|\{u,v\}\cap S|=1} w_S^* = d_{uv}\}$. That is, U consists of every node $u \in V\backslash\{v\}$ for which the length of the line segment joining u to v is completely accounted for by moats and disks. Since some pair uv for $u \in U$ is in M^*, we must have $U \neq \varnothing$. Suppose that there were distinct maximal members S^1 and S^2 of $\mathcal{S}'$ containing members u^1 and u^2 of U, respectively. Then

$$\begin{aligned} d_{u^1u^2} &\le d_{u^1v} + d_{u^2v} \qquad \text{(by the triangle inequality)} \\ &< r_{u^1}^* + r_{u^2}^* + \sum_{S\in\mathcal{S}',|S\cap\{u^1,v\}|=1} w_S^* + \sum_{S\in\mathcal{S}',|S\cap\{u^2,v\}|=1} w_S^* \\ &= r_{u^1}^* + r_{u^2}^* + \sum_{S\in\mathcal{S}',|S\cap\{u^1,u^2\}|=1} w_S^*. \end{aligned}$$

The last equality holds because S^1 and S^2 are maximal members of $\mathcal{S}'$ and v belongs to no member of $\mathcal{S}'$. However, this contradicts (3.3), i.e., feasibility of r^* and w^*. Therefore there must be a unique maximal member S of $\mathcal{S}'$ which contains a member u of U. (We may have $S = \{u\}$.) Since $d_{uv} \ge 0$, we must have $w_S^* > 0$.

Now increase r_v^* by ε and decrease w_S^* by ε until one of three things happens:

(i) r_v^* reaches zero, in which case we can continue to fix up another member v' of V having $r_{v'} < 0$, if one exists.
(ii) w_S^* becomes zero, in which case we delete S from $\mathcal{S}'$ and repeat the process.
(iii) Some new node u' enters U.

However, in the last case we would immediately find two maximal members of $\mathcal{S}'$ containing nodes of U, which, as we have seen, would contradict feasibility.

Therefore we can repeat this process, each time reducing the size of $\mathcal{S}'$ until we raise r_v^* to zero. Repeating, we can obtain equivalent r^* and w^*, such that $w^*, r^* \geq 0$. Note moreover that the changes we make do not change the value $\sum_{v \in V} r_v^* + \sum_{S \in \mathcal{Q}} w_S^*$.

Now we construct the moats and disks as follows. For each $v \in V$, construct a disk D_v of radius r_v^* centered at v. For each $S \in \mathcal{S}$, construct a moat M_S surrounding the nodes of S as follows. For each $v \in S$ define $\rho_v = r_S^* + \sum_{v \in T \subset S, T \in \mathcal{S}} w_T^*$ and let $\bar{\rho}_v = \rho_v + w_S^*$. Let S_v be a disk of radius ρ_v centered at v and let $\bar{S}_v$ be a disk of radius $\bar{\rho}_v$ centered at v. Then the moat M_S is defined as $M_S = \bigcup_{v \in S} \bar{S}_v \setminus \bigcup_{v \in S} S_v$, see Figure 3.4. The width of the moat is just w_S^*.

Feasibility of r^*, w^* ensures that the disks and moats thereby constructed will be nonoverlapping. Optimality of M^*, r^*, and w^* ensures that the length of M^* equals exactly the sum of the disk radii plus the moat widths. □

Now we describe a fast heuristic method for constructing a disk and moat packing, which will give us a lower bound. It is based on the minimum-cost spanning-tree algorithm.

Choose an arbitrary node $\hat{v}$ and consider the following pair of dual linear programs:

(LP) $$\max \sum_{\hat{v} \notin S} 2 w_S$$

(3.4) $$\sum_{S \subseteq V, |S \cap \{u,v\}| = 1} w_S \leq d_{uv} \qquad \text{for all } u, v \in V, \quad u \neq v,$$

(3.5) $$\sum_{S \subseteq V, v \in S} w_S = \alpha \qquad \text{for all } v \in V,$$

$$w_S \geq 0 \qquad \text{for all } S \subseteq V,$$

$$\alpha \text{ unrestricted.}$$

(DLP) $$\min \sum_{u, v \in V, u \neq v} d_{uv} x_{uv}$$

(3.6) $$x(\delta(S)) + y(S) \geq \begin{cases} 2 & \text{if } \hat{v} \notin S, \\ 0 & \text{if } \hat{v} \in S, \end{cases} \qquad \text{for all } S \subseteq V,$$

$$y(V) = 0,$$

$$x_{uv} \geq 0 \qquad \text{for all } u, v \in V, \quad u \neq v,$$

$$y_v \text{ unrestricted} \qquad \text{for all } v \in V.$$

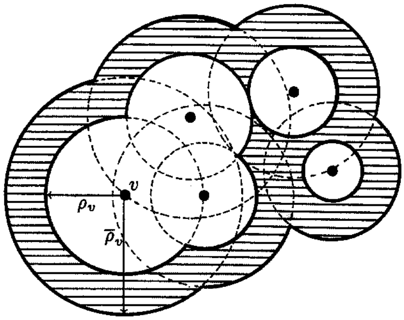

Fig. 3.4. Moat construction.

Problem (LP) is a variant on the disk/moat packing problem we have studied in this section. The differences are that:

(i) We can construct moats surrounding any set of nodes, not just odd sets.
(ii) We must "balance" the packing in that (3.5) requires that the sum of the widths of the sets of moats surrounding each node be equal.
(iii) The objective function ignores the moats surrounding one node $\hat{v}$ (the choice of which does not matter by (ii)), but doubles the rest.

Surprisingly, problem (DLP) is just the minimum-cost spanning-tree problem, slightly disguised. Kruskal's algorithm with minor modifications will build optimum solutions to both linear programs.

First, consider (LP). Recall Kruskal's algorithm builds a minimum-cost spanning-tree by starting with the shortest edge and then adding successively the shortest remaining edge which produces no cycle until $n - 1$ edges are chosen. When a tree edge t is added, it connects nodes in two trees T_1 and T_2 previously connected.

We build a solution w to (LP) as we go, starting with $w = 0$. At each stage, for each tree T that we have built, all nodes v of T will satisfy $\sum_{v \in S} w_S - \alpha(T) = 0$ where $\alpha(T)$ equals one-half of the length of a longest edge of T, respectively $\alpha(T) = 0$ if T has only one node.

Now suppose we add edge t joining nodes $u^1 \in T_1$ and $u^2 \in T_2$. We construct a moat of width $\frac{1}{2}d_{u^1u^2} - \alpha(T_i)$ around T_i for $i = 1, 2$. (Since $d_{u^1u^2}$ is at least as great as the length of the longest edge in T_1 (resp. T_2) these widths are nonnegative.) Let $\alpha(T) = \frac{1}{2}d_{u^1u^2}$, where T is the new tree produced. It is clear that when we terminate, w satisfies (3.4) and (3.5) with α equal to one-half of the length of a longest edge of T, the minimum-cost spanning tree produced. Note that the solution w satisfies (3.4) with equality for every edge of T.

Now we construct a feasible solution to (DLP). Let $x_{uv} = 0$ if uv is not an edge of T and let $x_{uv} = 1$ if uv is an edge of T. Choose arbitrarily some node $\hat{v}$ of T. Orient all edges of T toward $\hat{v}$. For each $v \in V$, define y_v equal to the outdegree of v (in T) minus its indegree. Using induction on T, we can prove that this is a feasible solution to (DLP). Since $x_{uv} = 1$ only for edges in T, for every such u, v (3.4) holds with equality. Again, using induction, we can show that (3.6) holds with equality for all $S \subseteq V$ with $w_S > 0$. (Show that it holds for two node trees, start with an arbitrary tree, remove one pendent node, apply induction.)

Therefore, these solutions satisfy the complementary slackness conditions for optimality, and we have the following.

THEOREM 3.2. *The optimal solutions to* (DLP) *are incidence vectors of minimum-cost spanning trees with node variables* (y_v) *equal to outdegrees minus indegrees, after all edges have been oriented toward an arbitrary root.*

Of particular interest to us is the solution to (LP), for it provides a feasible solution to the disk/moat packing associated with the minimum-length perfect-matching problem.

THEOREM 3.3. *Let T be a minimum-cost spanning tree of length $d(T)$ and let w^*, $\alpha(T)$ be an optimum solution of the linear program* (LP), *constructed during the running of Kruskal's algorithm. Then every perfect matching has length at least* $\frac{1}{2}d(T) + \alpha(T) - \sum_{|S|\text{even}} w_S^*$.

PROOF. If we set $w_S^* = 0$ for all S such that $|S|$ is even, then w_S^* is a feasible set of moat widths. Thus, if M^* is a minimum-length perfect matching, then

$$\begin{aligned} d(M) &\geq \sum_{|S|\text{odd}} w_S^* \\ &= \sum_S w_S^* - \sum_{|S|\text{even}} w_S^* \\ &= \sum_{v \notin S} w_S^* + \sum_{v \in S} w_S^* - \sum_{|S|\text{even}} w_S^* \\ &= \tfrac{1}{2}d(T) + \alpha(T) - \sum_{|S|\text{even}} w_S^*. \end{aligned}$$ □

Note that we can compute w^* and $\alpha(T)$ without changing the runtime of Kruskal's algorithm, hence this bound can be calculated very quickly.

Figure 3.5 shows an example on 10 nodes. We display a minimum-length spanning tree along with an optimum dual solution where the r_i^* are the radii of the circles around the nodes and the w_S^* are the width of the moats around node sets. In this example, we have

$$d_{14} = 20, \quad d_{25} = 30, \quad d_{34} = 40, \quad d_{45} = 10,$$
$$d_{57} = 70, \quad d_{68} = 20, \quad d_{78} = 40, \quad d_{7,10} = 30, \quad d_{89} = 10,$$

and therefore

$$r_1 = 10, \quad r_2 = 15, \quad r_3 = 20, \quad r_4 = 5, \quad r_5 = 5,$$
$$r_6 = 10, \quad r_7 = 15, \quad r_8 = 5, \quad r_9 = 5, \quad r_{10} = 15,$$
$$w_{\{4,5\}} = 5, \quad w_{\{1,4,5\}} = 5, \quad w_{\{1,2,4,5\}} = 5,$$
$$w_{\{1,2,3,4,5\}} = 15, \quad w_{\{7,10\}} = 5, \quad w_{\{8,9\}} = 5,$$
$$w_{\{6,8,9\}} = 10, \quad w_{\{6,7,8,9,10\}} = 15.$$

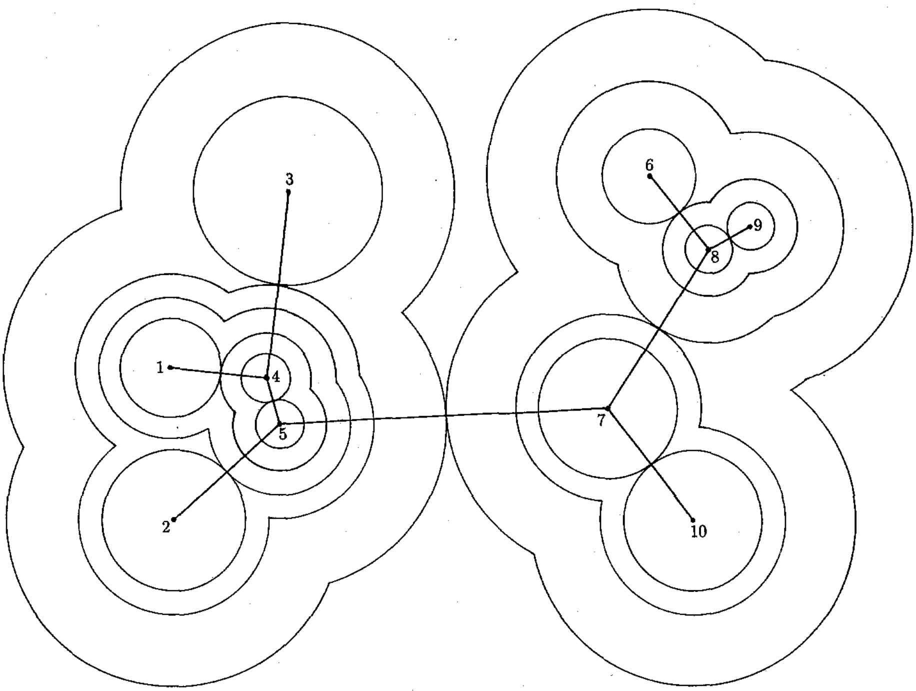

Fig. 3.5. Minimum spanning tree and moat packing.

The minimum spanning tree has length 270 and the lower bound for the minimum matching we get from Theorem 3.3 is 150.

Figure 3.6 displays a minimum-length matching along with all radii and all moats for odd cardinality node sets. (The w_S for the even cardinality node sets are set to zero.) We have $c_{12} = 32$ and $c_{36} = 76$, so the matching has length 158. The lower bound of 150 proves that the matching is at most about 5% off the optimum, and there is a gap of 2 on the edge (1, 2) and a gap of 6 on the edge (3, 6) in the dual solution. Without losing dual feasibility, we can increase

r_2	to	16
$w_{\{1,4,5\}}$	to	6
$w_{\{1,2,3,4,5\}}$	to	18
$w_{\{6,7,8,9,10\}}$	to	18

to establish the complementary slackness conditions also for the matching, which gives an optimality proof as displayed in Figure 3.7.

It remains an open problem to find an efficient way of doing this kind of improvement of the lower bound. Given a set of n nonoverlapping disks and moats in the plane, is it possible in, say $O(n \log n)$ time to increase some disks and moats in such a way that they become maximal, i.e., every one is in a tight distance constraint?

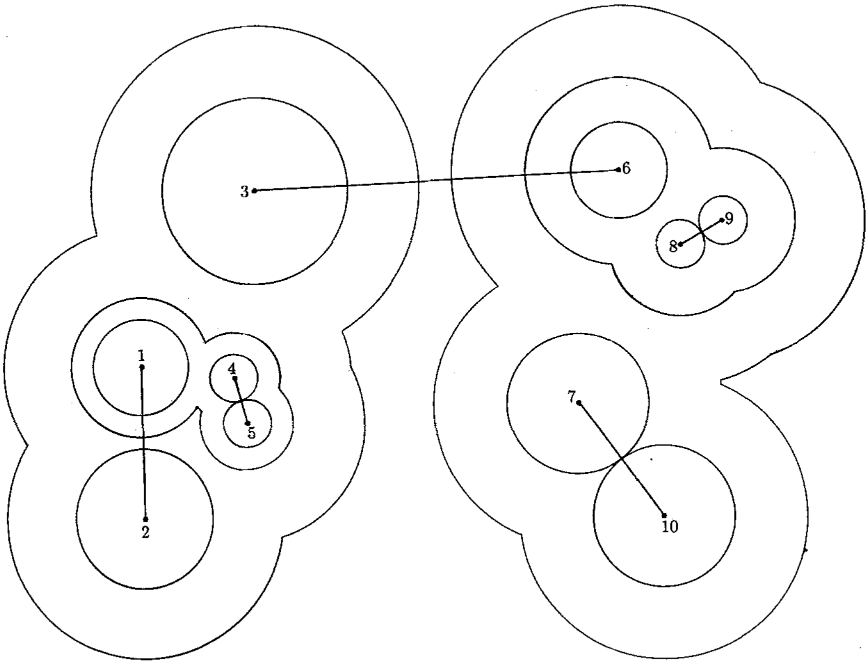

Fig. 3.6. Minimum matching and spanning tree bound.

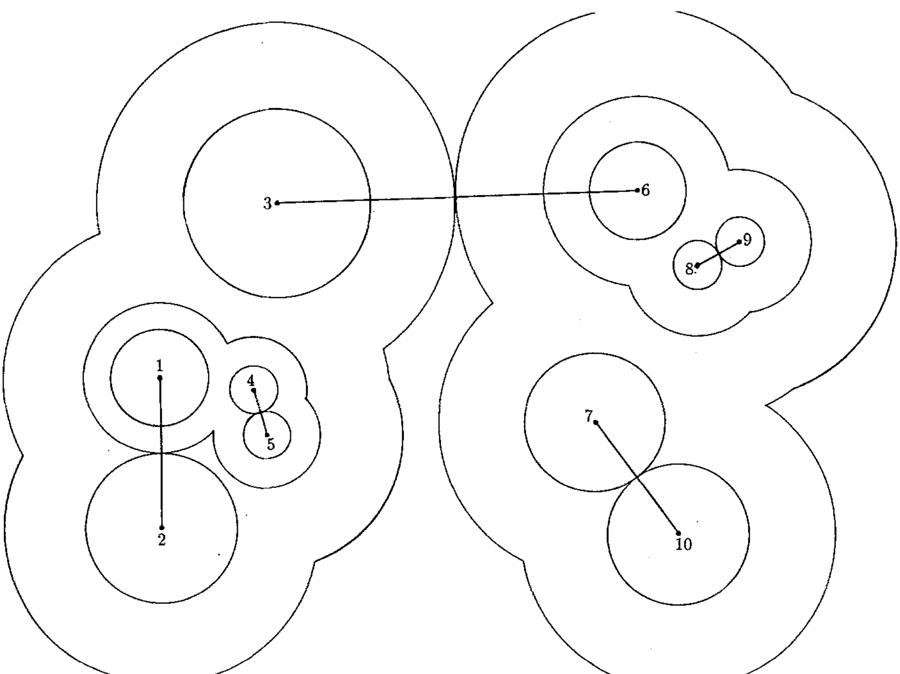

Fig. 3.7. Minimum matching and improved bound.

Finally, note that although we have used geometric intuition to explain the lower bound, it is completely general. The bound of Theorem 3.3 is valid for all objective functions.

In Jünger and Pulleyblank (1993) we give a survey of applications of these disk and moat packing ideas to a variety of combinatorial optimization problems. Recently, Goemans and Williamson (1992) showed that these methods could be extended to obtain an $O(n^2)\log n$ 2-approximation algorithm for perfect-matching problems whose costs satisfy the triangle inequality.

4. Computational Results. We have implemented the DUST heuristic along with the individual quality guarantee procedure described in Section 4 as a Pascal program. The Delaunay triangulation is computed with the incremental quaternary bucketing algorithm by Ohya *et al.* (1984). The spanning tree is computed with Kruskal's algorithm on the Delaunay graph using a heap as the priority queue and Tarjan's fast union-find technique. Very little code is added in this part to obtain the lower bound as described in Section 3. Finally, the matching is computed as described in Section 2.

Using our computer implementation we made several computational experiments on a Sun SPARCstation1+.

In a first set we generated random problem instances with n points chosen uniformly and independently in the unit square for $n = 1000, 2000, \ldots, 10{,}000$, ten instances of each size. The running times are displayed in Figure 4.1. We show minima, maxima, and averages. The problem instances are still small enough to make the timing behavior appear linear. As outlined in the Introduction, we

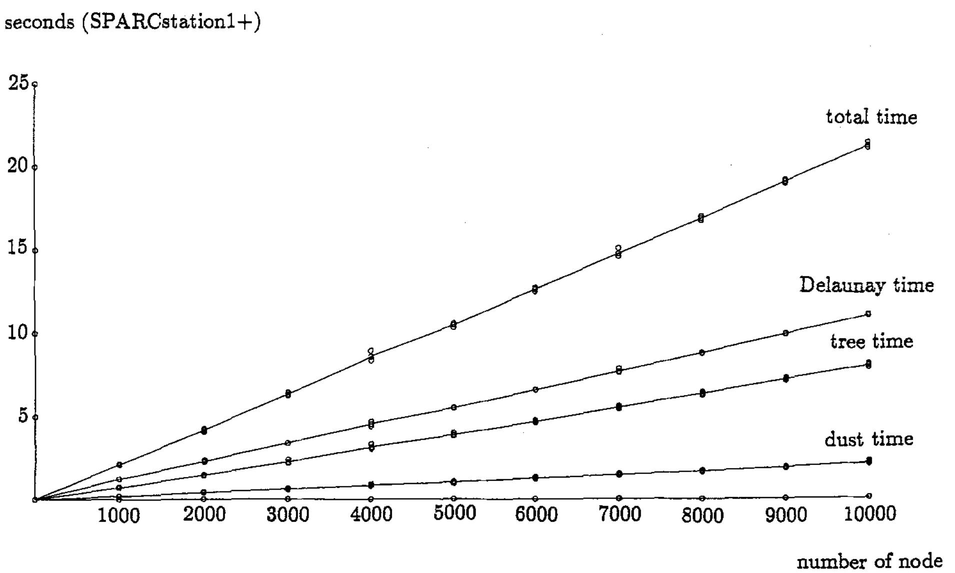

Fig. 4.1. Time on random problems.

obtained solutions which are about 6% above the optimum and our individual quality guarantee gives only "22% above the optimum." Both numbers had very small standard deviations.

This property seems not to be shared by real-world instances, where the data points are "grid-like." As a typical example, we consider Grötschel's 442-city instance of the traveling salesman problem, for which we computed the matching shown in Figure 4.2, which we can guarantee to be at most 6% off the optimum, and which is, in fact, only 1.2% off. The total computation time was 0.95 second.

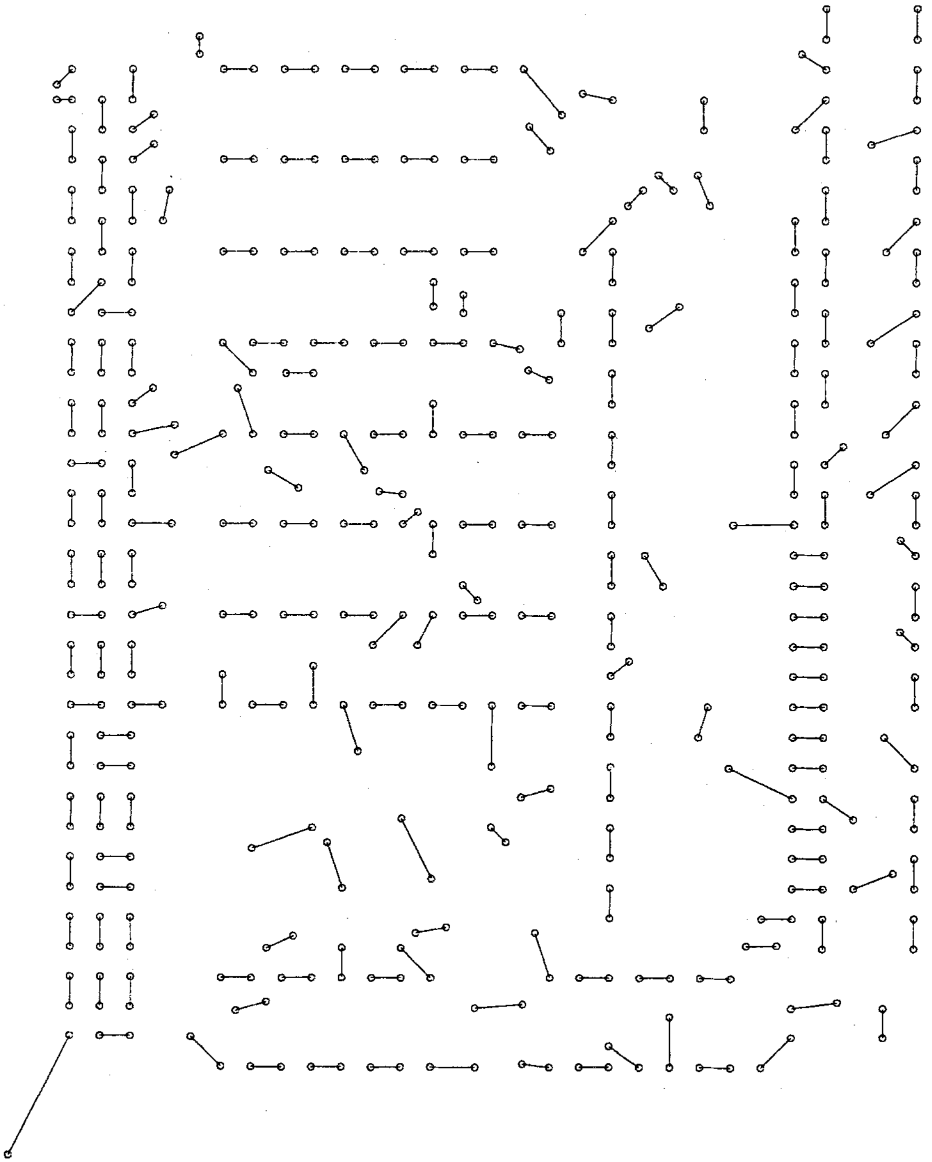

Fig. 4.2. 442-city problem.

Fig. 4.3. Street map of Kanto district.

Finally, we repeated a study made by Asano *et al.* (1985) for the Kanto district map of Japan. The task is to plot the map of the main streets shown in Figure 4.3 on a mechanical plotter in as little time as possible. If the street network on the map were connected, this would amount to finding a minimum-length perfect matching on all points of the map where an odd number of streets intersect or a street ends. The graph as shown, augmented by the matching edges, is then Eulerian, and the pen, starting and ending at any point inside the drawing, would traverse an Eulerian cycle, lifted up whenever traversing matching edges.

Unfortunately, the main street network on the map is not connected. In fact, there are 15 connected components, a very big one, a small one corresponding to Oshima island, and 13 very small ones, one for another small island and 12 on the borders of the map. The plotting problem for such unconnected maps can be reduced to a rural postman problem which is NP-hard. It can still be solved in polynomial time for a connected map plus an origin outside the map where the pen starts and ends its tour (see Grötschel *et al.*, 1991).

In our case we computed the shortest 15 "artificial streets" connecting the big component to the small ones, including the origin outside the map. After that, the map is connected and we have 4026 odd degree points defining our matching instance. Our program computed the solution shown in Figure 4.4 in 8.66 seconds. This matching is 7.4% above the optimum which was computed by W. Cook in about 6 minutes on a Sun SPARCstation1 + (personal communication) with a new implementation of Edmonds's blossom-shrinking algorithm for Euclidean instances.

We should point out here that in this real-world application the time the pen needs to move between two points on the plane is much better approximated if we measure the distance of the two points in Maximum (L_∞) metric rather than

Fig. 4.4. DUST matching.

Euclidean metric as done here. Our current implementation cannot handle this. (However, an L_∞ implementation should run even faster, because the running time is dominated by the Delaunay graph computation which is easier in this case.) Using our solution, we have measured the "lifted-up" pen movement, including the 15 artificial streets, in L_∞ metric, and get 6.52 m if the plot is 60 cm $\times$ 60 cm as described in Asano *et al.* (1985), where a 13.62-m solution was obtained using the "serpentine rack" matching heuristic.

Acknowledgments. We would like to thank Doris Zepf of the University of Augsburg for coding and debugging the quaternary bucketing algorithm for the Delaunay triangulation, Bill Cook of Bellcore for solving the 4026-point matching problems to optimality, Masao Iri and Kokichi Sugihara of the University of Tokyo for providing the data of the Kanto district map, and Petra Mutzel of the University of Cologne for careful reading of a preliminary version.

References

Aho, A. V., Hopcroft, J. E., and Ullman, D. U. (1974). *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, MA, 1974.

Asano, T., Edahiro, M., Imai, H., Iri, M., and Murota, K. (1985). Practical use of bucketing techniques in computational geometry. In: G. T. Toussaint (ed.), *Computational Geometry*, North-Holland, Amsterdam, 1985, pp. 153–195.

Avis, D. (1983). A survey of heuristics for the weighted matching problem. *Networks*, **13** (1983), 475–493.

Beardwood, J., Halton, J. H., and Hammersley, J. M. (1959). The shortest path through many points. *Proceedings of the Cambridge Philosophical Society*, **55** (1959), 299–327.

Christofides, N. (1976). Worst case analysis of a new heuristic for the travelling salesman problem. Technical Report, Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh, PA, 1976.

Cornuéjols, G., and Nemhauser, G. L. (1978). Tight bounds for Christofides' traveling salesman heuristic. *Mathematical Programming*, **14** (1978), 116–121.

Cunningham, W. H., and Marsh, A. (1978). A primal algorithm for optimum matching. *Mathematical Programming Study*, **8** (1978), 50–72.

Derigs, U. (1986). Solving large scale matching problems efficiently—a new primal matching approach. *Networks*, **16** (1986), 1–16.

Edmonds, J. (1965a). Paths, trees and flowers. *Canadian Journal of Mathematics*, **17** (1965), 449–457.

Edmonds, J. (1965b). Maximum matching and a polyhedron with 0–1 vertices. *Journal of Research of the National Bureau of Standards*, **69B** (1965), 125–130.

Goemans, M. X., and Williamson, D. P. (1992). A general approximation technique for constrained forest problems. *Proceedings of the Third Annual ACM–SIAM Symposium on Discrete Algorithms*, Association for Computing Machinery, New York, 1992, pp. 307–316.

Grigoriadis, M. D., and Kalantari, B. (1985a). A lower bound to the complexity of Euclidean and rectilinear matching algorithms. Technical Report LCSR-TR-69, Department of Computer Science, Rutgers University, New Brunswick, NJ, 1985.

Grigoriadis, M. D., and Kalantari, B. (1985b). A new class of heuristic algorithms for weighted perfect matching. Technical Report LCSR-TR-76, Department of Computer Science, Rutgers University, New Brunswick, NJ, 1985.

Grötschel, M., and Holland, O. (1985). Solving matching problems with linear programming. *Mathematical Programming*, **33** (1985), 243–259.

Grötschel, M., Jünger, M., and Reinelt, G. (1991). Optimal control of plotting and drilling machines: a case study. *ZOR—Methods and Models of Operations Research*, **35** (1991), 61–84.

Iri, M., Murota, K., and Matsui, S. (1983). Heuristics for planar minimum weight perfect matchings. *Networks*, **13** (1983), 67–92.

Jünger, M., and Pulleyblank, W. R. (1993). Geometric duality and combinatorial optimization. In: S. D. Chatterji, B. Fuchssteiner, U. Kulisch, and R. Liedl (eds.), *Jahrbuch Überblick Mathematik* 1993, Vieweg, Braunschweig/Wiesbaden, 1993, pp. 1–24.

Jünger, M., Reinelt, G., and Zepf, D. (1991). Computing correct Delaunay triangulations. *Computing*, **47** (1991), 43–49.

Mehlhorn, K. (1984). *Data Structures and Algorithms* 3: *Multi-Dimensional Searching and Computational Geometry*. Springer-Verlag, Berlin, 1984.

Ohya, T., Iri, M., and Murota, K. (1984). Improvements of the incremental method for the Voronoi diagram with computational comparison of various algorithms. *Journal of the Operations Research Society of Japan*, **27** (1984), 306–337.

Papadimitriou, C. H. (1977). The probabilistic analysis of matching heuristics. *Proceedings of the* 15*th Allerton Conference on Communication, Control and Computing*, 1977, pp. 368–378.

Reingold, E. M., and Tarjan, R. E. (1981). On a greedy heuristic for complete matching. *SIAM Journal on Computing*, **10** (1981), 676–681.

Shamos, M. I. (1975). Geometric complexity. *Proceedings of the* 7*th Annual ACM Symposium on Theory of Computing*, 1975, pp. 224–233.

Shamos, M. I., and Hoey, D. (1975). Closest point problems. *Proceedings of the* 16*th IEEE Annual Symposium on the Foundations of Computer Science*, 1975, pp. 151–162.

Sugihara, K. (1988). A simple method for avoiding numerical errors and degeneracy in Voronoi diagram construction. Research Memorandum RMI 88-14, Faculty of Engineering, University of Tokyo, 1988.

Sugihara, K., and Iri, M. (1988). Geometric algorithms in finite precision arithmetic. Research Memorandum RMI 88-10, Faculty of Engineering, University of Tokyo, 1988.

Supowit, K. J., Plaisted, D. A., and Reingold, E. M. (1980). Heuristics for weighted perfect matching. *Proceedings of the* 12*th Annual ACM Symposium on Theory of Computing*, 1980, pp. 398–419.

Tarjan, R. E. (1983). Data structures and network algorithms. Society for Industrial and Applied Mathematics, Philadelphia, PA, 1983.

Weber, M., and Liebling, T. M. (1985). Euclidean matching problems and the Metropolis algorithm. Technical Report, Département de Mathématiques, EPF Lausanne, 1985.