HEURISTICS FOR WEIGHTED PERFECT MATCHING[†]

Kenneth J. Supowit
David A. Plaisted
Edward M. Reingold

Department of Computer Science
University of Illinois
Urbana, Illinois 61801

ABSTRACT

The problem of finding near optimal perfect matchings of an even number n of vertices is considered. When the distances between the vertices satisfy the triangle inequality it is possible to get within a constant multiplicative factor of the optimal matching in time $O(n^2 \log K)$ where K is the ratio of the longest to the shortest distance between vertices. Other heuristics are analyzed as well, including one that gets within a logarithmic factor of the optimal matching in time $O(n^2 \log n)$. Finding an optimal weighted matching requires $\Theta(n^3)$ time by the fastest known algorithm, so these heuristics are quite useful.

When the n vertices lie in the unit (Euclidean) square, no heuristic can be guaranteed to produce a matching of cost less than $\frac{1}{\sqrt{\sqrt{12}}}\sqrt{n}$ in the worst case. We analyze various heuristics for this case, including one that always produces a matching costing at most $\frac{1}{\sqrt{2}}\sqrt{n}$. In addition, this heuristic also finds a traveling salesman tour of the n vertices costing at most $\sqrt{2}\sqrt{n}$. A different one of the heuristics analyzed produces asymptotically optimal results. It is also shown that asymptotically optimal traveling salesman tours can be found in $O(n \log n)$ time in the unit square.

INTRODUCTION

Consider the problem of finding a minimum cost matching in a weighted complete undirected graph G whose edges satisfy the triangle inequality. Let n, even, be the number of vertices in G. The most efficient algorithm known for the general weighted matching problem requires $\Theta(n^3)$ time, and we would like to find good approximation algorithms for the special case of the triangle inequality and the special case of the vertices lying in the unit (Euclidean) square. The former case was first considered in Reingold and Tarjan [14] and they analyzed the behavior of a greedy heuristic; the latter case was first considered by Papadimitriou [10] who was concerned with the expected cost of a matching.

Motivation for studying this approximation problem is threefold: First, as described in [14], matching has direct applications to minimizing the time required to draw networks on a mechanical plotter; in such cases the $\Theta(n^3)$ optimizing algorithm is unacceptable since n can be large. Second, a sufficiently close approximation to an optimal matching could be used to improve Christofides' traveling salesman problem heuristic [3], [4] without really harming the closeness of its approximation. Finally, matching is an interesting combinatorial problem in its own right and as such its approximation is also of interest.

We will consider two similar, but not identical, versions of the matching problem, each of which corresponds to a physical situation. First, we consider the general case of matching when the weights satisfy the triangle inequality. The results we obtain here are also applicable to our more specialized second case, that of n points in a bounded region of the Euclidean plane (typically the unit square). In the case of the bounded region (motivated by the plotter application referred to above) we will analyze a heuristic's behavior by bounding the absolute cost of the matching found, irrespective of the cost of an optimal matching. In the case of the triangle inequality (that is, an unbounded region) the cost of the matching can be unboundedly large for any number of vertices and so we must consider a measure of how bad the heuristically found match is compared to the optimal match, namely the ratio of the two costs.

TRIANGLE INEQUALITY

Let G be a complete undirected graph with

† This research was supported in part by the National Science Foundation, grant numbers NSF MCS 77-22830 and NSF MCS 79-04897.

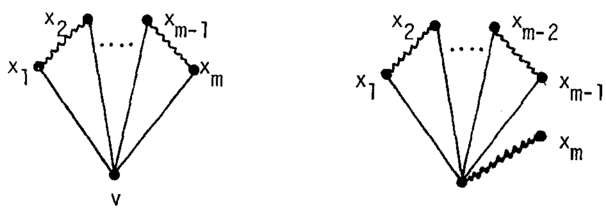
n vertices and weighted edges satisfying the triangle inequality. Let OPT(G) denote the minimum cost of a matching of G. Let M(G) be the cost of a matching produced by algorithm M. Let $R_M(n)$ be the worst case ratio M(G)/OPT(G) as a function of n, the number of vertices of G.

In [14], Reingold and Tarjan considered the greedy heuristic (GR) that repeatedly matches the two closest unmatched points. This can be implemented in worst case time $O(n^2 \log n)$, a significant improvement over the optimizing algorithm. The closeness of the approximation, however, is not very satisfactory. Reingold and Tarjan showed that

$$R_{GR}(n) \approx \frac{4}{3} n^{\lg\frac{3}{2} - 1} \approx \frac{4}{3} n^{.585}$$

and that this bound is achievable for all n.

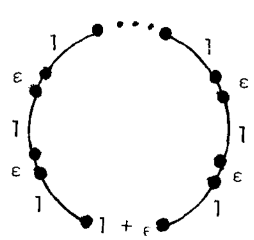
Papadimitriou [12] proposed an $O(n^2)$ heuristic based on spanning trees (ST): Begin with spanning tree on the vertices and convert it into a matching by replacing "flowers" $x_1, x_2, \ldots, x_m$, v in the tree by matching vertices as indicated by the wavy lines:



m even m odd

Then all vertices matched and all edges incident on them are deleted from the tree and the process is repeated. Papadimitriou showed that the ratio of the cost of the matching thus found to that of the optimal matching can be as bad an $\frac{n}{2}$ and no worse. We present an independently found proof here.

That the $\frac{n}{2}$ ratio is asymptotically achievable follows from Papadimitriou's example



In this example, the optimal match obviously consists of $\frac{n}{2} - 1$ edges of length ε and one edge of length $1 + \varepsilon$ with total cost $1 + \frac{n}{2}\varepsilon$, while the heuristic produces a matching with $\frac{n}{2}$ edges of length 1 for a total cost of $\frac{n}{2}$. Thus

$$R_{ST}(G) \geq \frac{n/2}{1 + \frac{n}{2}\varepsilon}$$

which approaches $\frac{n}{2}$ as $\varepsilon \to 0$.

To prove that $R_{ST}(n) \leq \frac{n}{2}$, suppose we are given a minimum spanning tree. We partition the edges of the tree into two classes

Even = {e | removal of e results in two subtrees each of which contains an <u>even</u> number of vertices}

Odd = {e | removal of e results in two subtrees each of which contains an <u>odd</u> number of vertices}

(Recall that n, the number of vertices, is even.) The desired result follows directly from three claims.

<u>Claim 1</u>: $ST(G) \leq \sum_{e \in Odd} cost(e)$

<u>Proof</u>: Immediate from the triangle inequality since by its nature the heuristic chooses only edges of Odd or edges whose cost is bounded above by the sum of two edges of Odd. <u>QED Claim 1</u>

<u>Claim 2</u>: Let t be the maximum number of odd edges on any path in the minimum spanning tree. Then,

$$\sum_{e \in Odd} cost(e) \leq t \cdot OPT(G)$$

<u>Proof</u>: If an edge e of the optimal matching is not in the minimum spanning tree, then adding e to the tree causes a cycle in which each edge has cost at most cost(e) (see [13]). If the cycle has edges from Odd of costs $c_1, c_2, \ldots, c_m$ then $c_i \leq cost(e)$ and summing this we get $\Sigma c_i \leq m \cdot cost(e)$. Summing this inequality over all edges e of the optimal matching we get on the right a value that is at most $t \cdot OPT(G)$ where t is as defined above. On the left we get a value that is at least $\sum_{e \in Odd} cost(e)$ (i.e., every odd edge appears on the left at least once) because every vertex in each of the two sets of odd cardinality is matched in the optimal match and at least one must be matched to a vertex in the other set. The claim follows. <u>QED Claim 2</u>

<u>Claim 3</u>: $t \leq \frac{n}{2}$, where t is as defined in Claim 2.

<u>Proof</u>: Define a mapping from vertices to edges of the tree as follows: Let the path consist of vertices $v_1, v_2, \ldots, v_k$ (in order). For $i = 2, 3, \ldots, k$ (in that order) map to the edge

(v_{i-1}, v_i) both v_i and all vertices that are disconnected from v_i by the removal of the edge (v_{i-1}, v_i) and that have not been previously mapped to some v_j, $j < i$. This mapping maps some of the n vertices to the edges of the path, and it follows easily by induction and the nature of an odd edge that each edge from Odd on the path is the image of at least two different vertices. Since there are only n vertices in the tree, it follows that if the path has o edges from Odd then $2o \leq n$ and $o \leq \frac{n}{2}$ as desired. QED Claim 3

Putting these claims together yields

$$ST(G) \leq \sum_{e \in Odd} cost(e) \leq t \cdot OPT(G) \leq \frac{n}{2} \cdot OPT(G),$$

so that $R_{ST}(G) \leq \frac{n}{2}$.

We now present two heuristics, the hyper-greedy (HG) heuristic and the factor of two (F2) heuristic. We show that $R_{HG}(n) \approx 2\log_3 n$ and $R_{F2}(n) \leq 8$. A refinement of the factor of two heuristic, the factor of two with sorting (F2S), gives $R_{F2S}(n) \leq 7$. To lower bound the worst case ratio, we have found graphs G with arbitrarily many vertices such that $HG(G)/OPT(G) \approx 2\log_3 n$. As with the spanning tree heuristic, these graphs are embedded in the circumference of the unit circle. Also, we have found graphs with arbitrarily many vertices demonstrating that $R_{F2}(n) > 4 - \varepsilon$ and $R_{F2S}(n) > 3 - \varepsilon$. By slightly simplifying the heuristics, we obtain the hyper-greedy heuristic without bridges and the factor of two heuristic without bridges. These have ratios at least as large as about $n^{\log_3(3/2)} \approx n^{.369}$ and $n^{\log_2(5/4)} \approx n^{.322}$, respectively. The graphs achieving these ratios can be embedded in a line, as with the bad examples for the greedy algorithm in [14]. Therefore, the use of bridges is an essential part of these heuristics. The hyper-greedy heuristic runs in time $O(n^2 \log n)$. The factor of two heuristic runs in time $O(n^2 \log K)$, where K is the ratio of the largest to the smallest edge weights in G, and is never worse than $O(n^3)$. The factor of two heuristic with sorting runs in time $O(n^2(\log n + \log K))$ and is never worse than $O(n^3)$. The hyper-greedy heuristic without bridges runs in time $O(n^2)$, and the factor of two heuristic without bridges runs in time $O(n^2 \log K)$, and is never worse than $O(n^3)$. If G is sparse, and weights os missing edges are taken to be the length of the shortest path between the endpoints, then the hyper-greedy heuristic runs in time $O(E \log^2 n)$ where G has E edges. The factor of two heuristic runs in time $O(E \log n \log K)$ in this case. These heuristics can be modified to solve the following problem, for an arbitrary weighted graph G not necessarily satisfying the triangle inequality: Find a low cost subgraph G' of G such that every node in G appears in G' and has odd degree in G'. The heuristics have the same asymptotic running time and performance bounds for this problem as for the weighted matching problem.

The basic idea of the heuristics is to collapse subsets of the nodes of G into "supernodes" to obtain a graph G_1. The heuristic is then applied recursively to G_1 to obtain a subgraph G' of G_1. Also, a spanning tree is constructed within each supernode of G_1, and the flower heuristic (see above) is applied to obtain a matching of a subset of this spanning tree. This is done so that these matchings, when combined with G', yield a subgraph of G in which every node has odd degree. This subgraph is then converted to a matching by repeatedly applying the triangle inequality.

It is necessary to distinguish "odd vertices" of G and "even vertices" of G for this to work. Supernodes of G_1 are constructed entirely from odd vertices of G. A supernode having an odd number of elements is called an odd supernode, and one having an even number of elements is called an even supernode. Also, even vertices of G are considered even supernodes of G_1. The graph G' is constructed so that odd supernodes have odd degree and even supernodes have even degree. The matchings within supernodes are constructed to match nodes of even degree in G'. Note that each supernode will have an even number of such vertices. The final result is a subgraph of G in which odd vertices have odd degree and even vertices have even degree. To start the heuristics, all vertices are considered odd vertices.

The Hyper-greedy Method

The hyper-greedy method works in the following way: Suppose $G = (V, E)$ is the given undirected graph satisfying the triangle inequality. We construct a sequence $G_0, G_1, G_2, \ldots, G_k$ of graphs as follows: G_0 is G. Let G_i be (V_i, E_i) in general (thus V_i are the vertices of G_i and E_i are the edges). Also, $V_i = Odd_i \cup Even_i$, $Odd_i \cap Even_i = \emptyset$, where Odd_i are the "odd vertices" of G_i and $Even_i$ are the "even vertices" of G_i. We have $Odd_0 = V$ and $Even_0 = \emptyset$. Let P_i be a set of paths in G_i connecting odd vertices with odd vertices of G_i. We choose P_i so that the sum of the weights of the paths in P_i is as small as possible, subject to the condition that each odd vertex of G_i is connected to one of its nearest odd neighbors by a path in P_i. A "nearest odd neighbor" of v is an odd node w which can be reached from v by a path in G_i of minimal length. It will turn out that G_i need not satisfy the triangle inequality for $i \geq 1$, so a path from v to w may have length smaller than the length $d_i(v, w)$ of the edge between v and w in G_i. We will show below how P_i may be efficiently computed using a "generalized Voronoi diagram".

Let G_i' be the graph (V_i', E_i') where E_i' is the set $\{\{v, w\}$: there is a path in P_i having v and w as endpoints$\}$. It will turn out that G_i' consists of a disjoint collection of trees, plus isolated vertices (the vertices in $Even_i$). A connected component of G_i' having an odd number of vertices, at least 3 vertices, is called an <u>odd component</u> of G_i'. A connected component having an even number of vertices is called an <u>even component</u> of G_i'. A connected component having a single vertex is an element of $Even_i$ and is considered to be an even component of G_i'.

Note that every odd component of G_i' will have at least 3 vertices. Hence $|Odd_{i+1}| \leq \frac{1}{3}|Odd_i|$. The sequence $G_0, G_1, \ldots, G_k$ stops when $Odd_k = \emptyset$. Since $|Odd_i|$ is even for all i, $k \leq \log_3(3n/2)$.

An edge between V1 and V2 in G_i corresponds to an edge between v1 and v2 in G_{i-1}, for some $v1 \in V1$ and $v2 \in V2$ such that $d_{i-1}(v1, v2)$ is minimal. Similarly, an edge in G_{i-1} corresponds to an edge in G_{i-2}. Continuing in this way, an edge in G_i corresponds to an edge in G. Also, every edge in G_i' corresponds to a path in G_i, and therefore to a set of edges in G_i, hence a set of edges in G. We keep track of these correspondences between edges of G_i', edges of G_i, and edges of G to construct a matching of G.

We obtain a matching by examining the graphs $G_k, G_{k-1}, \ldots, G_0$ in order. We first use the "flower heuristic" on all the trees of G_{k-1}' to obtain matching of the odd vertices of G_{k-1}. (Recall that $Odd_k = \emptyset$ so G_k has no odd vertices.) Each tree edge in G_{k-1}' corresponds to a path in P_{k-1}, hence to a path in G_{k-1}. The flower heuristic matches vertices in a tree by edges or pairs of edges from the tree. By applying the flower heuristic, we obtain a set of paths in G_{k-1} matching the odd vertices of G_{k-1}. The actual edges in G are obtained from these edges in G_{k-1} as indicated above. We then use the flower heuristic on G_{k-2}', passing over the nodes which are endpoints of the paths in G_{k-1}. By the way paths are constructed, an even number of vertices will already be matched in each even tree and an odd number of vertices will be matched in each odd tree. Hence each tree in G_{k-2}' will have an even number of vertices remaining to be matched. Thus the flower heuristic yields a match on G_{k-2}', and we interpret each edge of this matching as a set of edges of G as before. We then proceed to G_{k-3}, using the flower heuristic but passing over vertices which have been matched in G_{k-1}' or G_{k-2}', and so on.

To analyze the worst case ratio, let T_i be the total length of the trees at level i. Let H_i be the total length of the match edges produced by this heuristic at level i. Let M_i be the total length of the optimal edges at level i. (We assign levels to optimal edges by grouping them into "paths" between vertices of G_i for various i.) We have $T_i \leq 2M_i$ by a simple argument except that vertices of T_i may have been matched at levels higher than i. Therefore we have $T_i \leq 2M_i + 2M_{i+1} + \ldots + 2M_k$ for all i. Summing over i, noting that $H_i \leq T_i$ for all i, we get that $\sum_{i=0}^{k-1} H_i \leq 2k \sum_{i=0}^{k-1} M_i$. (Note that $H_k = M_k = 0$.) Since $k \leq \log_3(1.5n)$, we have a ratio bounded by $2\log_3(1.5n)$.

<u>The Factor of 2 Method</u>

The factor of 2 method is similar to the hyper-greedy method except that paths of P_i are included in a different manner. Let ℓ be the length of the shortest path between odd vertices of G_i; then P_i includes all paths between odd vertices of G_i whose length is in the interval $(\ell, 2\ell)$. However, paths occurring in cycles are deleted until P_i consists of a set of disjoint trees. Other than this, the factor of 2 method is identical to the hyper-greedy method. Note that we cannot guarantee $k \leq \log_3 (1.5n)$ in this case. Instead, $k \leq \log_2 K$.

The analysis is similar to that of the hyper-greedy method, except that $T_i \leq 4M_i$ ignoring vertices matched at a higher level. Including these, and noting that edges at higher levels get longer and longer, we have that $T_i \leq 4M_i + 2M_{i+1} + M_{i+2} + \ldots$. Summing over i, we obtain that $\sum_i T_i \leq 8\sum_i M_i$ so the ratio is at most 8.

The factor of 2 heuristic with sorting differs in that paths with length in the range $(\ell, 2\ell)$ are included in order of size, skipping over paths that would form cycles with paths already included in P_i. Thus we construct a set of "minimum spanning trees" of the components of G_i'. We now have $T_i \leq 3M_i$ except for vertices matched at a higher level. Including these, we get $T_i \leq 3M_i + 2M_{i+1} + M_{i+2} + \frac{1}{2}M_{i+3} + \ldots$ so $\sum_i T_i \leq 7\sum_i M_i$, giving a ratio of at most 7.

The heuristics without bridges are the same except P_i only includes paths of length 1 (that is, single edges). In other words, we consider the distance between odd vertices to be the length of the edge connecting them.

Implementations

We construct the graphs G_i for the three bridge heuristics using generalized Voronoi diagrams, as follows:

Given a graph G and a subset S of the vertices of G, the generalized Voronoi diagram for G relative to S is defined as a partition of the vertices of G according to which element of S they are closest to. Associated with each vertex v of S we have a Voronoi region consisting of all vertices of G that are closer to v than to any other element of S. (Ties may be broken arbitrarily.) Also, with each vertex of G we keep the distance to the closest element of S. Since G may fail to satisfy the triangle inequality, this distance is the length of the shortest path to an element of S. It is not difficult to see that the generalized Voronoi diagram can be constructed in $O(n^2)$ time if G has n vertices. If G is sparse, the Voronoi diagram can be constructed in O(E log n) time.

We obtain G_{i+1} from G_i for the hyper-greedy method using the generalized Voronoi diagram as follows: Let VG_i be the generalized Voronoi diagram of G_i relative to Odd_i. It turns out that if $v \in Odd_i$ and w is the closest odd vertex of G_i to v then the Voronoi regions of v and w will be adjacent. That is, there will be an edge in G_i connecting a vertex in the Voronoi region of v with a vertex in the Voronoi region of w. Therefore, by examining all edges in G_i whose endpoints lie in different Voronoi regions, we can find the sets P_i and E_i'. This requires time proportional to the number of edges of G_i. Finally, constructing $G_{i+1} = G_i/E_i'$ given G_i and E_i' requires time proportional to the number of edges in G_i. Therefore each step $G_i \rightarrow VG_i \rightarrow E_i' \rightarrow G_i/E_i'$ takes $O(n^2)$ time and the work per level is $O(n^2)$ for a total of $O(n^2 \log_3 n)$. For sparse graphs, $O(E(\log n)^2)$ suffices.

The generalized Voronoi diagram also suffices for the factor of 2 methods with and without sorting, for the following reason: If v and w are odd vertices of G_i then the Voronoi regions of v and w in VG_i will be adjacent unless there is an odd vertex x of G_i such that $d_i(v, x) \leq d_i(v, w)$ and $d_i(w, x) \leq d_i(v, w)$. To see this, consider a shortest path between v and w in G_i. If some vertex on this path is not in the Voronoi region of v or w, then this vertex must be in the Voronoi region of some vertex x as above. Therefore, if v and w may be connected by a path of length 2ℓ or less, then v and x may be connected by such a path, and x and w may be connected by such a path. Hence v and w will still end up in the same component of G_i' if the Voronoi diagram is used to construct the components.

The number of levels for the factor of two heuristic is bounded by $\lceil \lg K \rceil$ since the edge length doubles each time. However, the number of levels may be much less than this, and will never be larger than n. Hence the total work for the factor of two heuristic is $O(n^2 \log K)$ and is never more than $O(n^3)$. Possibly this heuristic can be implemented more efficiently than this. For sparse graphs, O(E logn logK) time suffices.

The factor of two heuristic with sorting requires the sorting of edges and paths. Although there may be many levels, whenever two edges or paths must be compared it means that there will be fewer odd vertices and paths in later levels. The total sorting time is therefore $O(n^2 \log n)$. The construction of minimum spanning trees can be done using the UNION-FIND algorithm [13], which takes negligible time. Since there may be log K levels, the work to construct generalized Voronoi diagrams is $O(n^2 \log K)$. The total work is therefore $O(n^2(\log n + \log K))$. For sparse graphs, O(E logn logK) suffices.

The hyper-greedy heuristic without bridges runs in time $O(n^2)$ since the number of odd vertices is a decreasing geometric series. For the factor of two heuristic without bridges, $O(n^2 \log K)$ work suffices since there are up to log K levels. It would be interesting to know if better heuristics exist that run in $O(n^2)$ time. Also, is there a heuristic with a constant worst-case ratio that runs in time $O(n^2 \log n)$?

BOUNDED EUCLIDEAN REGIONS

Here we will measure the performance of a heuristic by the absolute cost of the matching produced in the unit square. If we have n points in the unit square then no heuristic can do better than $\frac{1}{\sqrt{\sqrt{12}}}\sqrt{n} \approx .537\sqrt{n}$ in the worst case, since that is the cost of the optimal matching if n points on a 1 by 1 hexagonal grid. In fact, we will be able to come close to this bound.

Avis [2] has analyzed the greedy heuristic on the unit square. He has shown that a matching thus found will have cost at most $\frac{2}{\sqrt{\sqrt{12}}}\sqrt{n} \approx 1.07\sqrt{n}$, although the worst known case has cost $\frac{3}{2\sqrt{\sqrt{12}}}\sqrt{n} \approx .806\sqrt{n}$. This performance is poor, especially considering that the algorithm requires time proportional to $n^2 \log n$. In the results below we will improve dramatically on both the cost of the matching and the time required.

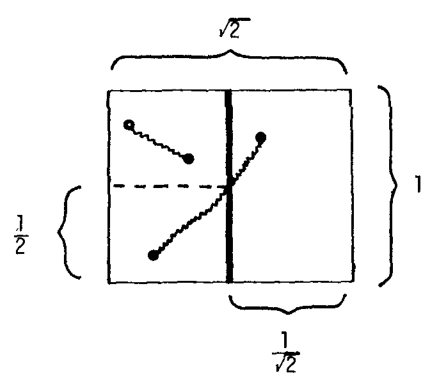
Partition Algorithms

Here we present a class of $O(n \log n)$ time algorithms, each of which operates by partitioning the region into subregions and recursively solving the smaller matching problems thus obtained. If a subregion contains an odd number of points, then all but one are matched and the odd point is then matched with an odd point in another subregion (there must be another since there is an even number of points in total).

The first of these algorithms we consider is the rectangle heuristic, which works as follows. The unit square is imagined to be enclosed in a $\sqrt{2}$ by 1 rectangle. If $n \geq 2$ then this rectangle is split into two equal-sized subrectangles, each having a $\sqrt{2}$ to 1 ratio between the long and the short sides. The algorithm is performed recursively on each of the two subrectangles. In general, when called on a rectangle R, the algorithm does the following:

if R contains ≥ 2 input points,
then 1. split R into two rectangles R_1 and R_2 each having a $\sqrt{2}$ to 1 ratio between the long and short sides
2. perform the algorithm on R_1
3. perform the algorithm on R_2
4. if R_1 and R_2 each contain an odd number of input points
then
put the edge (p_1, p_2) in the matching, where p_1 is the input point in R_1 which was not matched in step 2, and p_2 is that of R_2 not matched in step 3.

As an example, in the figure below n = 4:



The first split was on the heavy solid line. The left half was then split along the dotted line. The matching produced is in jagged line.

There is one more detail of the algorithm: the level of recursion is not allowed to go beyond $\lceil \lg n \rceil$. More precisely, define a rectangle to be either the main $\sqrt{2}$ by 1 rectangular region, or one of two rectangular subregions with sides having ratio $\sqrt{2}$ to 1 into which a rectangle may be split. Also, let R(P) denote the subset of P contained in rectangle R. Furthermore, if R is a rectangle, then let

$$\text{level}(R) = \begin{cases} 0, & \text{if } R \text{ is the main } \sqrt{2} \text{ by 1 rectangle} \\ \text{level}(R') + 1, & \text{otherwise, where } R' \text{ is a rectangle which splits into } R \text{ and some other rectangle} \end{cases}$$

The algorithm now is:

if level(R) $\leq \lceil \lg n \rceil$
then do as described above
else arbitrarily match up the input points in R until 0 or 1 is left

The reason for this restriction on the depth of recursion is that it enables the algorithm to run in time $O(n \log n)$. The time is dominated by the partitioning of the points. Now for each rectangle R, for each input point $p \in R(P)$, we can decide with a single comparison which half of R p lies in. Also, for each input point p, we make at most 1 of these comparisons on each level of recursion, and hence at most $\lceil \lg n \rceil$ such comparisons in total. Hence the time is $O(n \log n)$.

In order to analyze the performance, that is the worst case cost of the matching produced by the algorithm, we first find that worst cost for arbitrary sets of points in the $\sqrt{2}$ by 1 rectangle. Later, we will use this result to upper bound the cost for a set of points all in a 1 by 1 square within the $\sqrt{2}$ by 1 rectangle.

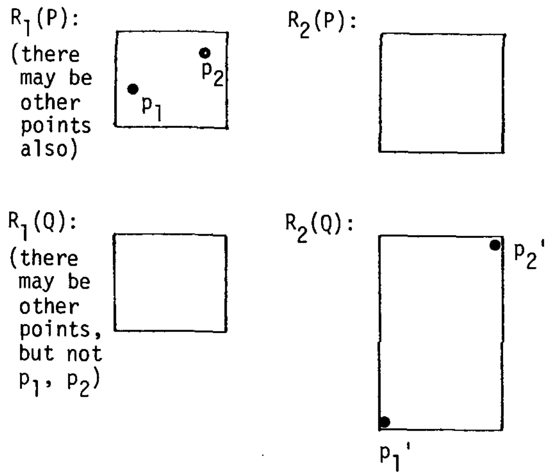
If P is a set of points in the $\sqrt{2}$ by 1 rectangle, then let rcost(P) denote the sum of the lengths of the edges in the matching produced by the rectangle algorithm on P. For all $n \geq 0$, let $C_n = \sup\{\text{rcost}(P): P \text{ is a set of } n \text{ points}\}$. By "set of points" we mean, here and throughout this section, a set of points in the $\sqrt{2}$ by 1 rectangle. Note that we are not primarily interested in C_n for odd n; they are defined so as to help analyze C_n for even n. Our first lemma shows that the restriction to $\lceil \lg n \rceil$ levels of recursion does not affect the C_n.

Lemma 1: Let $n > 0$, P a set of n points. Then $(\forall$ set of points $Q)[|Q| = n$ and $\text{rcost}(Q) \geq \text{rcost}(P)$ and no level $\lceil \lg n \rceil + 1$ rectangle contains ≥ 2 points of Q].

Proof: First, we introduce some notation used throughout the analysis. If P' is a set of points, and R a rectangle, then let R(P') denote the set of points of P' within R.

Now if $(\forall$ level $\lceil \lg n \rceil + 1$ rectangle R) $[|R(P)| \leq 1]$, then we have nothing to prove. So let R_1 be a level $\lceil \lg n \rceil + 1$ rectangle such that $|R_1(P)| \geq 2$. Then $R_2(P)$ is empty for some level $\lceil \lg n \rceil$ rectangle R_2, for otherwise $|P| \geq 2^{\lceil \lg n \rceil} + 1$

> n (since there are $2^{\lceil lgn \rceil}$ level $\lceil lgn \rceil$ rectangles). Our strategy now is to show that the points of P can be rearranged to produce a set Q of n points such that $rcost(Q) \geq rcost(P)$ and $|R_1(Q)| = |R_1(P)| - 2$ and $|R_2(Q)| = 2$, but otherwise Q is just like P. Let $p_1, p_2 \in R_1(P)$ such that p_1 is matched to p_2 by the algorithm. Define Q to be just like p except that $p_1, p_2 \notin Q$ and Q has points p_1' and p_2' in opposite corners of R_2. Thus:



Now it is easily proved by induction on i that the dimensions of a level i rectangle are $\frac{\sqrt{2}}{(\sqrt{2})^i}$ by $\frac{1}{(\sqrt{2})^i}$. $\therefore$ the length of a long diagonal in a level i rectangle is $\frac{\sqrt{3}}{(\sqrt{2})^i}$.

$$\therefore \quad d(p_1, p_2) \leq \frac{\sqrt{3}}{(\sqrt{2})^{\lceil lgn \rceil + 1}} < \frac{\sqrt{3}}{(\sqrt{2})^{\lceil lgn \rceil}}$$

$$= d(p_1', p_2') .$$

This "moving" of the two points into R_2 does not affect the algorithm's matching of the other points. $\therefore$ $rcost(Q) > rcost(P)$. In this manner we continue to rearrange P until no level $\lceil lgn \rceil + 1$ rectangle has ≥ 2 points in it. QED Lemma 1.

From here on, we analyze the algorithm as if there were no restriction on the depth of recursion. Lemma 1 implies that this assumption does not affect the worst case costs, that is, the C_n.

Our strategy is to define a class of input sets and then show that these sets are the worst case for the algorithm. Specifically, we say that a set of points P is balanced if for all rectangle R such that $|R(P)| \geq 2$, R splits into rectangles R_1, R_2 such that

(i) if 4 divides $|R(P)|$ then $|R_1(P)| = \frac{|R(P)|}{2} - 1$ and $|R_2(P)| = \frac{|R(P)|}{2} + 1$.

(ii) if 4 does not divide $|R(P)|$ then $|R_1(P)| = \left\lfloor \frac{|R(P)|}{2} \right\rfloor$, $|R_2(P)| = \left\lceil \frac{|R(P)|}{2} \right\rceil$.

(iii) If $|R(P)|$ is even then the point p_1 stranded (i.e. left unmatched) by the call on R_1 and the point p_2 stranded by R_2 are in opposite corners of R.

Note that we do not require $|P|$ to be even; we define balanced sets of odd cardinality in order to help analyze those of even cardinality. In other words, for a balanced set, each rectangle R with an even non-zero number of points splits odd-odd, with the two subrectangles having almost the same number of points, and the edge produced at the end of the call on R is along one of R's diagonals. Intuitively, one might expect such a set P to be a worst case for the algorithm. This is indeed the case, as is proved in the next two lemmas.

Lemma 2: Let $n \geq 0$ be even, and P a set of n points. Then ($\exists$ set of points Q)[$|Q| = n$ and $rcost(Q) \geq rcost(P)$ and ($\forall$ rectangle R such that $|R(Q)| \geq 1$

[1. $|R(Q)|$ even $\Rightarrow$ R splits into R_1, R_2 such that $|R_1(Q)|$, $|R_2(Q)|$ are odd, and R_1 and R_2 strand points of Q in opposite corners of R,

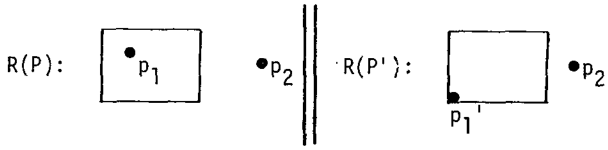
2. $|R(Q)|$ odd $\Rightarrow$ R strands a point of Q in one of its own corners,

3. $|R(Q)| \geq 2 \Rightarrow$ the two subrectangles of R each contain at least 1 point of Q]].

(When we say a rectangle R' "strands" an input point p we mean that p is within R' and is not matched by the algorithm to another point in R').

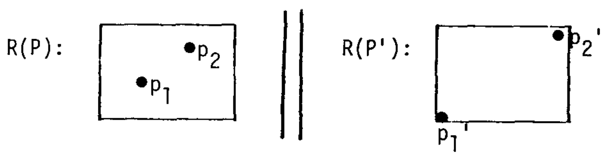
Proof: We will rearrange P (in the manner of lemma 1) so as to satisfy the desired property, and then will let Q be this new P.

First we consider all rectangles R such that $|R(P)| = 1$. Let R be such a rectangle, and let p_1 be the point in R(P). Since n is even, the algorithm must match p_1 to some other point $p_2 \in P$ outside of R. If p_1 is already in a corner of R, then define P' to be like P except that instead of having p_1, P' has point p_1' in the corner of R which is farthest from p_2. Thus,



Hence $d(p_1, p_2) < d(p_1', p_2)$. Since this "moving" of p_1 to p_1' affects no other matches made by the algorithm on P, we have rcost(P) < rcost(P'), and $|P'| = |P| = n$. Thus, we let P be P' and continue with the rearranging.

Having so rearranged, if necessary, all rectangles containing exactly 1 point of P, we now consider those containing 2 points. Let R be such a rectangle, $R(P) = \{p_1, p_2\}$. Since $|R(P)|$ is even, the arrangement of the points of P within R does not affect the matching of any points outside of R. Therefore if p_1, p_2 are not in opposite corners of R, then "move" them there by letting P' be like P except that instead of having p_1 and p_2, P' has p_1' and p_2' in opposite corners of R, thus

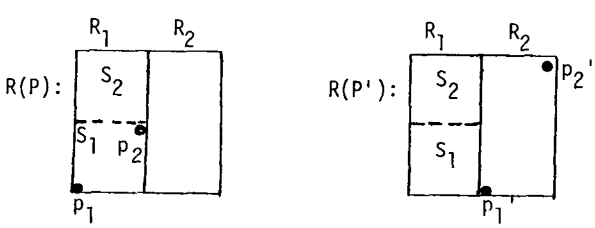


Since $d(p_1, p_2) < d(p_1', p_2')$, we have rcost(P) < rcost(P'), $|P'| = |P|$, which is what we want; so let P = P'.

Now assume we have rearranged all rectangles R such that $|R(P)| \leq k$ for some integer $k \geq 2$. We will now rearrange each rectangle R such that $|R(P)| = k+1$. Let R be such a rectangle.

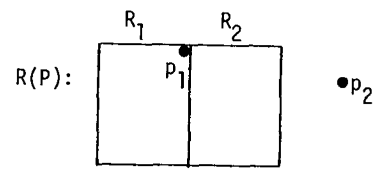
<u>Case 1</u>: k + 1 is odd. Then R splits into rectangles R_1, R_2 such that $|R_1(P)|$ is odd and $|R_2(P)|$ is even.

<u>Case 1.1</u>: $|R_2(P)| = 0$. Then $|R_1(P)| \geq 3$. $\therefore$ R_1 splits into some rectangles S_1, S_2 such that $|S_1(P)| \geq 2$. Let p_1, p_2 be two points in S_1 matched to each other by the algorithm (such points must exist since S_1 strands at most one point and if S_1 strands one point then $|S_1(P)| \geq 3$). Now define P' to be exactly like P except that P' has points p_1' and p_2' in opposite corners of R_2, and no point at p_1 or p_2. Thus,

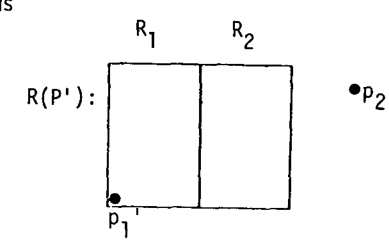


Moving p_1 and p_2 out of S_1 does not affect the matching of the other points in R_1. Also, $d(p_1, p_2) < d(p_1', p_2')$. $\therefore$ rcost(P) < rcost(P') and $|P| = |P'|$, so let P = P' and continue to rearrange R. That is, $|R_1(P)|$ is now < k+1. $\therefore$ rearrange R_1, and then rearrange R, using case 1.2 below. (This procedure terminates since $|R_1(P)| < |R(P)|$).

<u>Case 1.2</u>: $|R_2(P)| > 0$. Then $|R_1(P)|$, $|R_2(P)| <$ k+1 and hence both R_1 and R_2 have already been rearranged. In particular, R_1 strands a point p_1 in a corner of R_1. The algorithm matches p_1 to some point p_2 outside of R. If p_1 is already in a corner of R, then we have nothing to rearrange. So assume p_1 is not in a corner of R. Thus, e.g.



Now let P' be like P except that the points in R_1 has been rotated and perhaps swapped with those in R_2 so that p_1 is now in an extreme corner from p_2. Thus



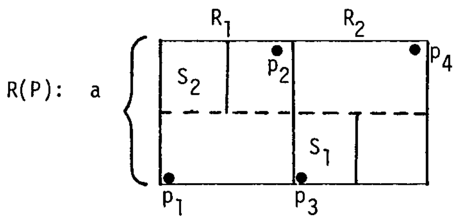
This rotating and swapping has no effect on the cost of the matching of the points in R(P) other than p_1. $\therefore$ rcost(P) < rcost(P'), and since $|P'| = |P|$, let P = P' and continue with the rearranging.

<u>Case 2</u>: k + 1 is even. Let R_1, R_2 be the subrectangles of R, and assume, without loss of generality, that $|R_1(P)| \geq |R_2(P)|$.

<u>Case 2.1</u>: $|R_2(P)| = 0$. Then proceed exactly as in Case 1.1.

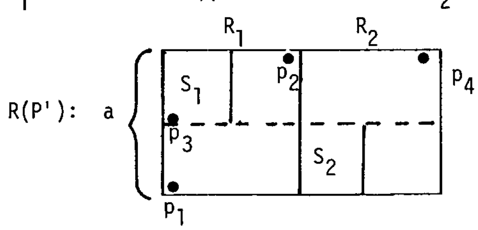
<u>Case 2.2</u>: $|R_2(P)| > 0$. Then $|R_1(P)|$, $|R_2(P)| <$ k+1. $\therefore$ R_1 and R_2 have already been rearranged. Since $|R(P)| = |R_1(P)| + |R_2(P)|$ is even, we have two cases:

<u>Case 2.2.1</u>: $|R_1(P)|$, $|R_2(P)|$ are both even. This is the most interesting of all the cases, since it is the only one which depends on the shape of our rectangles. Since R_1, R_2 already satisfy the desired properties, we have the following situation:



That is, R is a rectangle of size $a\sqrt{2}$ by a, for some $a > 0$. R_1, a subrectangle of R, matches points p_1 and p_2 in opposite corners of R_1. R_2 similarly matches p_3 and p_4 in its opposite corners. S_2 is the even subrectangle of the subrectangle of R_1 which strands p_2. S_1 is the odd subrectangle of the subrectangle of R_2 which strands p_3. (We say a rectangle R' is <u>even</u> if $|R'(P)|$ is even, otherwise it is <u>odd</u>).

Now let P' be like P except that the points in S_1 have been swapped with those in S_2:



Hence, $rcost(P) = d(p_1, p_2) + d(p_3, p_4) + c$ for some $c \geq 0$, and $rcost(P') = d(p_1, p_4) + d(p_2, p_3') + c$. Now $d(p_1, p_2) = d(p_3, p_4) = \sqrt{(\frac{a\sqrt{2}}{2})^2 + a^2} = \frac{a\sqrt{3}}{2}$. Also, $d(p_1, p_4) = \sqrt{a^2 + (a\sqrt{2})^2} = a\sqrt{3}$, and $d(p_2, p_3') = \sqrt{(\frac{a}{2})^2 + (\frac{a\sqrt{2}}{2})^2} = \frac{a\sqrt{3}}{2}$.

$$\therefore \quad rcost(P) = 2(a\,\frac{\sqrt{3}}{\sqrt{2}}) + c = a\sqrt{6} + c < \frac{a3\sqrt{3}}{2} + c$$
$$= a\sqrt{3} + \frac{a\sqrt{3}}{2} + c = rcost(P').$$

Hence, since $|P'| = |P|$, we have what we want, so let P = P' and continue to rearrange.

<u>Case 2.2.2</u>: $|R_1(P)|$, $|R_2(P)|$ are both odd. Since $|R_1(P)|$, $|R_2(P)| < k+1$, we already have that R_1 strands a point p_1 in one of its corners, and R_2 strands a point p_2 in one of its corners. If p_1 and p_2 are not in opposite corners of R, then the appropriate rotations of $R_1(P)$ and $R_2(P)$ will produce a set P' of cost greater than that of P.

Thus, we continue to rearrange P, until we have rearranged the main, level 0, rectangle. Then let Q be this final arrangement. Q satisfies the properties stated in the lemma. <u>QED Lemma 2</u>

The set Q constructed from P in Lemma 1 has some of the properties of a balanced set, but not all. The next lemma rearranges this Q so as to be balanced, without changing rcost(Q). This completes the claim that balanced sets constitute a worst case for the algorithm.

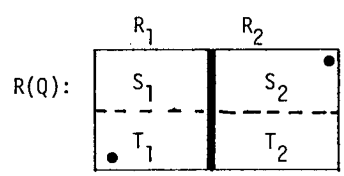
<u>Lemma 3</u>: Let $n \geq 0$ be even, P a set of n points. Then $(\exists$ set of points $Q_1)[|Q_1| = n$ and $rcost(Q_1) \geq rcost(P)$ and Q_1 is balanced].

<u>Proof</u>: Let Q be a set satisfying the properties stated in Lemma 2. We will rearrange Q to a new set Q_1 such that ($\forall$ rectangle R) [if R_1, R_2 are the 2 subrectangles of R then $||R_1(Q_1)| - |R_2(Q_1)|| \leq 2$]. Furthermore, Q_1 will still have the property of Lemma 2 that even, non-empty rectangles split odd-odd stranding points in opposite corners. Together, these properties imply that Q_1 is balanced.

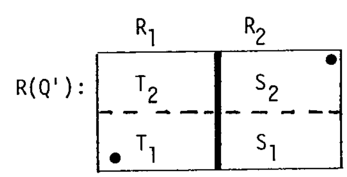
First, note that all rectangles R such that $|R(Q)| = 1$ or 2 are already balanced, and hence need no rearranging.

Assume we have balanced all rectangles R such that $|R(Q)| \leq k$ for some integer k. Let R be a rectangle such that $|R(Q)| = k+1$. Let R_1, R_2 be the subrectangles of R, and S_1, T_1 the subrectangles of R_1, and S_2, T_2 the subrectangles of R_2. Say that a rectangle R' is <u>even</u> if $|R'(Q)|$ is even, otherwise R' is <u>odd</u>.

<u>Case 1</u>: R is even. Then R_1, R_2 are odd, by our choice about Q. Assume WLOG that T_1, S_2 are odd, thus



Then swap $S_1(Q)$ with $T_2(Q)$, to get, in the notation of Lemma 2,



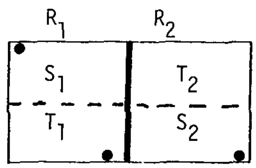
Since $|R_1(Q)|$, $|R_2(Q)| \leq k$, we have that R_1, R_2 were balanced before this swap. Therefore, letting $s_1 = |S_1(Q)|$, $s_2 = |S_2(Q)|$, $t_1 = |T_1(Q)|$, $t_2 = |T_2(Q)|$, we have that $|s_1 - t_1| = 1$ and $|s_2 - t_2| = 1$.

$\therefore \quad ||R_1(Q')| - |R_2(Q')|| =$

$|(t_1 + t_2) - (s_1 + s_2)| \leq 2$, which is what we want. Now this swapping of $S_1(Q)$ with $T_2(Q)$ may have made R_1 or R_2 (or both) unbalanced. $\therefore$ we now rearrange R_1 and R_2 (this procedure terminates since $|R_1(Q')|, |R_2(Q')| < |R(Q')|$). Thus R is now balanced, so we let $Q = Q'$ and continue to rearrange other rectangles.

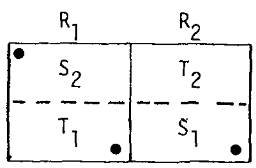
Case 2: R is odd. Assume, WLOG, R_1 is even and R_2 is odd. Define s_1, s_2, t_1, t_2 as in Case 1. By the choice of Q, $|R_1(Q)| > 0$ and hence $|R_1(Q)|$, $|R_2(Q)| \leq k$. $\therefore$ s_1, t_1 are odd. Assume WLOG, s_2 is odd and $s_1 \geq t_1$, thus

R(Q):



Case 2.1: $s_2 \geq t_2$. Then since R_2 is balanced, we have $s_2 = t_2 + 1$. Then swap $S_1(Q)$ with $S_2(Q)$ to get Q', thus

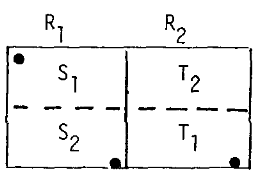
R(Q'):



Note that we also may need to rotate $S_2(Q)$ so that its stranded point is opposite that of T_1. Since $0 \leq s_1 - t_1 \leq 2$, we have $||R_1(Q')| - |R_2(Q')||$ $= |(s_2 + t_1) - (s_1 + t_2)| = |(s_2 - t_2) + (t_1 - s_1)|$ $= |1 + (t_1 - s_1)| \leq 1$, which is what we want.

Case 2.2: $s_2 < t_2$. Then $s_2 = t_2 - 1$. Swap $T_1(Q)$ with $S_2(Q)$ to get Q', thus (after possibly rotating)

R(Q'):



Now $||R_1(Q')| - |R_2(Q')|| = |(s_1 + s_2) - (t_1 + t_2)|$ $= |(s_2 - t_2) + (s_1 - t_1)| = |-1 + (s_1 - t_1)| \leq 1$, as desired.

Thus let $Q = Q'$, and after re-balancing R_1 and R_2 if necessary, continue to rearrange other rectangles.

Finally, after balancing the main, level 0, rectangle, let Q_1 be this new Q, and we are done. Note that the rearrangement can change neither the cost of the set, nor the assumed properties of Q. <u>QED Lemma 3</u>

Thus the balanced sets constitute the worst case for the algorithm; that is, for all even $n \geq 0$, $C_n = \text{rcost}(P)$, where $|P| = n$ and P is balanced. We now analyze the C_n.

$C_0 = C_1 = 0$, $C_2 = \sqrt{3}$, $C_3 = \frac{\sqrt{3}}{\sqrt{2}}$. A balanced set of 4n points splits into two balanced sets - one with 2n + 1 points, and one with 2n - 1 points - and matches 2 points in its opposite corners.

Thus $\forall n \geq 1$, $C_{4n} = \frac{1}{\sqrt{2}}(C_{2n+1} + C_{2n-1}) + \sqrt{3}$. The factor $\frac{1}{\sqrt{2}}$ is to scale down the cost from the $\sqrt{2}$ by 1 region to the 1 by $\frac{1}{\sqrt{2}}$ region. More precisely, the length of a longest edge on level i + 1 is $\frac{\sqrt{3}}{\sqrt{2}^{i+1}} = \frac{1}{\sqrt{2}}\left(\frac{\sqrt{3}}{\sqrt{2}^{i}}\right) = \frac{1}{\sqrt{2}}$(the length of a longest edge on level i).

Similarly, $\forall n \geq 1$

$$C_{4n+1} = \frac{1}{\sqrt{2}}(C_{2n+1} + C_{2n}),$$

and $\forall n \geq 0$,

$$C_{4n+2} = \frac{1}{\sqrt{2}}(C_{2n+1} + C_{2n+1}) + \sqrt{3},$$

$$C_{4n+3} = \frac{1}{\sqrt{2}}(C_{2n+2} + C_{2n+1}) .$$

For notational convenience, let $\alpha = \frac{1}{\sqrt{2}}$, and $D_n = \frac{1}{\sqrt{3}} C_n$ $\forall n \geq 0$. Then it can be shown by induction on i that for all $i \geq 1$, $D_{i+1} - D_{i-1}$ $= \alpha^{\lceil \lg(\frac{3}{4}i) \rceil}$.

We were not able to solve for each C_n exactly. We can however, put a rather tight upper bound on the C_n. Our strategy is to define a special class of n and then solve (to within an $O(\frac{1}{\sqrt{n}})$ term) for C_n for n in this class. Then we will show that this function of n uppers bounds C_n for all n.

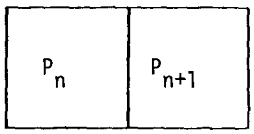
Given an integer $r \geq 0$, we say that a set P is <u>full to level r</u> if

(i) P is balanced
(ii) ($\forall$ rectangle R)
[1. level (R) $\leq r - 1 \Rightarrow |R(P)| > 0$, and
2. level (R) $\geq r \Rightarrow |R(P)| \leq 1$].

Note that this definition implies that every level r rectangle has 0 or 1 points of P in it, and every level r - 1 rectangle has 1 or 2 points of P in it.

We say that an integer n is <u>full to level r</u> if there exists a set P such that $|P| = n$ and P is full to level r. We now show that $(\forall r \geq 0)$ $(\exists n \geq 0)$ [n, n + 1 are both full to level r]. Now let $r \geq 0$ and assume that n and n + 1 are both full to level r. Then $\exists$ sets P_n, P_{n+1} such that $|P_n| = n$, $|P_{n+1}| = n + 1$ and P_n and P_{n+1} are both full to level r. Now we construct two sets both full to level r + 1:

<u>Case 1</u>: n is even. Then let P_{2n+1} be the set consisting of P_n in its left subrectangle, and P_{n+1} in its right subrectangle:



Also, let P_{2n+2} be the set consisting of P_{n+1} as its left subrectangle and P_{n+1} as its right subrectangle. Then both P_{2n+1} and P_{2n+2} are full to level r + 1.

<u>Case 2</u>: n is odd. Then let P_{2n} be the set with subrectangles consisting of P_n and P_n. Also, let P_{2n+1} be the set with subrectangles consisting of P_n and P_{n+1}. Then P_{2n}, P_{2n+1} are both full to level r + 1.

Thus 0, 1 are full to level 0, and if , $\ell + 1$ are full to level r then ℓ even $\Rightarrow 2\ell + 1$, $2\ell + 2$ are full to level r + 1, and ℓ odd $\Rightarrow 2\ell$, $2\ell + 1$ are full to level r + 1. Thus the sequence (0,1, 1,2, 2,3, 5,6, 10,11, 21,22, ...) consists of numbers full to some level. In fact, it is easily proved by induction that this sequence contains all numbers full to some level. Call the sequence the full numbers. Incidentally, it is also easy to show that if P is a balanced set, then (P is full to some level) $\Leftrightarrow$ ($\forall$ rectangle R such that $|R(P)| > 0$)[4 does not divide $|R(P)|$].

Now let $r \geq 0$ and P a set full to level r, such that $|P| = n$ is even. We wish to relate n and r. For all $i \geq 0$, let $E_i = |\{$rectangle R: level (R) = i and $|R(P)|$ is even and $\geq 2\}|$. Similarly, let $O_i = |\{$rectangle R: level (R) = i and $|R(P)|$ is odd$\}|$. Since n is even, we have that $E_0 = 1$, $O_0 = 0$. Since P is balanced, we have that each non-empty even rectangle splits odd-odd, and (of course) each odd rectangle splits odd-even. Thus,

$$\forall\, 1 \leq i \leq r-1, \quad O_i = O_{i-1} + 2E_{i-1}, \quad E_i = O_{i-1}.$$

Also, since P is full to level r, we have $E_i = 0$ $\forall i \geq r$. Also note that $\forall\, 0 \leq i \leq r-1$, $O_i + E_i = 2^i$ since there are a total of 2^i level i rectangles. The solution to this recurrance is $O_i = \frac{2}{3}(2^i - (-1)^i)$ for $0 \leq i \leq r-1$, and

$$E_i = \begin{cases} \frac{2}{3}(2^{i-1} - (-1)^{i-1}), & \text{for } 0 \leq i \leq r-1 \\ 0 & \text{, for } i \geq r. \end{cases}$$

Now since P is balanced, we can associate with each even, non-empty rectangle R a pair $\{p_1, p_2\} \subseteq P$ such that p_1 and p_2 are in opposite corners of R and are matched to each other by the algorithm. These $\frac{n}{2}$ pairs form a partition of P.

$$\therefore\ n = \sum_{i=0}^{r-1} 2 \cdot E_i = 2\Big[\sum_{i=0}^{r-1} \frac{2}{3}(2^{i-1} - (-1)^{i-1})\Big]$$
$$= \frac{2^{r+1}}{3} + \frac{2}{3}(-1)^{r+1}.$$

Define, for all $r \geq 0$, $b_r = \frac{2^{r+1}}{3} + \frac{2}{3}(-1)^{r+1}$. Then, as just shown, the sequence $(b_0, b_1, b_2, \ldots) = (0,2,2,6,10,22,42,\ldots)$ consists of all even full numbers. Also for all $r \geq 0$, let $w_r = \left\lfloor \frac{2^{r+1}}{3} \right\rfloor$. The sequence $(w_0, w_1, w_2, \ldots) = (0,1,2,5,10,21,\ldots)$ arises in connection with merge insertion (Knuth [8], p. 187) and with an algorithm for finding the greatest common divisor of two integers (Knuth [7], exercise 4.5.2 - 2.7). Knuth points out that it is curious that this sequence arises in such different settings. We now add to this list of curiosities by observing that

$$w_r = \begin{cases} \frac{2^{r+1}}{3} - \frac{2}{3} = b_r, \text{ if } r \text{ even} \\ \frac{2^{r+1}}{3} - \frac{1}{3} = b_r - 1, \text{ if } r \text{ odd.} \end{cases}$$

Thus, w_r is the smaller of the two numbers full to level r.

Now fix some $r \geq 0$, and some P full to level r such that $|P|$ is even (i.e., $|P| = n = b_r$). We analyze rcost(P), that is C_{b_r}.

$$\text{rcost}(P) = \sum_{i=0}^{r-1} E_i \cdot (\text{length of a long diagonal of a level } i \text{ rectangle})$$

$$= \sum_{i=0}^{r-1} \frac{2}{3}(2^{i-1} - (-1)^{i-1}) \cdot \frac{\sqrt{3}}{(\sqrt{2})^i}$$

$$= \frac{\sqrt{2}}{\sqrt{3}}\left(1 + \frac{1}{\sqrt{2}}\right)\sqrt{2}^r + \sqrt{3} - \sqrt{6}$$
$$+ \frac{\sqrt{2}}{\sqrt{3}}(2 - 2\sqrt{2})\left(-\frac{1}{\sqrt{2}}\right)^r.$$

Now $n = \frac{2^{r+1}}{3} + \frac{2}{3}(-1)^{r+1}$.

$$\therefore\quad r = \lg(\tfrac{3}{2}n) + O(\tfrac{1}{n}) \text{ (Using the Taylor expansion).}$$

$$\therefore\quad \sqrt{2}^r = \sqrt{2}^{\lg(\frac{3}{2}n) + O(\frac{1}{n})} = \sqrt{\frac{3n}{2}} \cdot \sqrt{2}^{O(\frac{1}{n})} =$$
$$\sqrt{\frac{3n}{2}}\left(1 + O(\tfrac{1}{n})\right) = \sqrt{\frac{3n}{2}} + O(\tfrac{1}{\sqrt{n}}).$$

Also, $(-\frac{1}{\sqrt{2}})^r = (-\frac{1}{\sqrt{2}})^{\lg(\frac{3n}{2}) + O(\frac{1}{n})} = O(\frac{1}{\sqrt{n}})$

$\therefore \quad C_n = \text{rcost}(P) = (1 + \frac{1}{\sqrt{2}})\sqrt{n} + \sqrt{3} - \sqrt{6} + O(\frac{1}{\sqrt{n}}).$

Thus we know (up to an $O(\frac{1}{\sqrt{n}})$ term) C_n for an infinite class of even n. Now we consider the other even values of n. Fix some $t \geq 0$. We wish to upper bound C_{2t}. Recall $D_{2t} = \frac{1}{\sqrt{3}} C_{2t}$. Let 2m be the largest integer such that $2m \leq 2t$ and $2m = b_k$ for some $k \geq 0$. Then we can write D_{2t} as

$$D_{2m} + \sum_{\substack{i \text{ odd} \\ 2m+1 \leq i \leq 2t-1}} (D_{i+1} - D_{i-1}) .$$

Recall that this implies

$$D_{2t} = D_{2m} + \sum_{\substack{i \text{odd} \\ 2m+1 \leq i \leq 2t-1}} \alpha^{\lceil \lg(\frac{3}{4}i) \rceil} .$$

Now as formulas (17) and (18) of Knuth [8], p. 187, imply that $(\forall w_k < i \leq w_{k+1})[\lceil \lg(\frac{3}{4}i)\rceil = k]$.

$\therefore$ in particular, $\lceil \lg(\frac{3}{4}i)\rceil = k \;\; \forall$ odd i such that $w_k \leq 2m < 2m+1 \leq i \leq 2t-1 < 2t \leq w_{k+1}$.

$$\therefore \sum_{\substack{i \text{ odd} \\ 2m+1 \leq i \leq 2t-1}} \alpha^{\lceil \lg(\frac{3}{4}n) \rceil} = (t - m)\alpha^k .$$

Next we express K in terms of m. Note that k is even $\Leftrightarrow w_k$ is even. $\therefore$ if k is even then

$w_k = 2m = \frac{2^{k+1}}{3} - \frac{2}{3}$ and hence $k = \lg(3m + 1)$.

If k is odd then $w_k = 2m-1 = \frac{2^{k+1}}{3} - \frac{1}{3}$ and hence $k = \lg(3m - 1)$. Thus

$$\begin{aligned} D_{2t} &= D_{2m} + (t-m)\alpha^k = D_{2m} + (t-m)(\tfrac{1}{\sqrt{2}})^k \\ &\leq D_{2m} + (t-m)(\tfrac{1}{\sqrt{2}})^{\lg(3m-1)} \\ &= \tfrac{1}{\sqrt{3}} C_{2m} + \tfrac{t-m}{\sqrt{3m-1}} \\ &= \tfrac{1}{\sqrt{3}}[(1 + \tfrac{1}{\sqrt{2}})\sqrt{2m} + \sqrt{3} - \sqrt{6} + O(\tfrac{1}{\sqrt{m}})] \\ &\quad + \tfrac{t-m}{\sqrt{3m-1}} \\ &= \tfrac{1}{\sqrt{3}}(\sqrt{2} + 1)\sqrt{m} + 1 - \sqrt{2} + O(\tfrac{1}{\sqrt{m}}) + \tfrac{t-m}{\sqrt{3m-1}}. \end{aligned}$$

<u>Lemma 4</u>: $\frac{1}{\sqrt{3}}(\sqrt{2} + 1)\sqrt{m} + 1 - \sqrt{2} + O(\frac{1}{\sqrt{m}}) + \frac{t-m}{\sqrt{3m-1}}$

$$\leq \frac{1}{\sqrt{3}}(1 + \sqrt{2})\sqrt{t} + 1 - \sqrt{2} + O(\frac{1}{\sqrt{t}}).$$

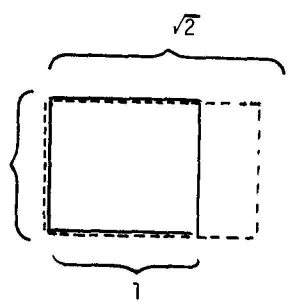
<u>Proof</u>: Let $d = \frac{1}{\sqrt{3}}(\sqrt{2} + 1)$. Since $2m \leq 2t < b_{k+1} \leq 4m+1$, we have that $O(\frac{1}{\sqrt{m}}) = O(\frac{1}{\sqrt{t}})$. $\therefore$ we need show only that $d\sqrt{m} + \frac{t-m}{\sqrt{3m-1}} \leq d\sqrt{t}$, i.e. that $d\sqrt{t} - d\sqrt{m} + \frac{t-m}{\sqrt{3m-1}} \geq 0$. Let 2r be the least even full number $\geq 2t$. $\therefore$ $m \leq t \leq r$. Define the function $f: [m, r] \to \mathbb{R}$ by $f(y) = d\sqrt{y} - d\sqrt{m} - \frac{y-m}{\sqrt{3m-1}}$.

Then $f'(y) = 0 \Leftrightarrow y = \frac{d^2(3m-1)}{4}$. Furthermore $f''(y) < 0 \;\; \forall m \leq y \leq r$. $\therefore$ f is minimized in the range [m, r] at m or at r. Now by the definitions of m and r, $t = m \Leftrightarrow t = r$. $\therefore$ $(\forall m \leq y \leq r)$ $[f(y) \geq f(m) = d\sqrt{m} - d\sqrt{m} - \frac{m-m}{\sqrt{3m-1}} = 0]$. <u>QED Lemma 4</u>

By Lemma 4, $D_{2t} \leq \frac{1}{\sqrt{3}}(1 + \sqrt{2})\sqrt{t} + 1 - \sqrt{2} + O(\frac{1}{\sqrt{t}})$. An argument similar to the above (but using $k = \lg(3m + 1)$ instead of $\lg(3m - 1)$ shows that $C_{2t} \geq (\frac{1}{2\sqrt{2}} - \frac{3}{2} + 2\sqrt{2})\sqrt{2t} + \sqrt{3} - \sqrt{6} - o(1) \approx 1.68\sqrt{2t} - .717 - o(1)$. We state the upper bound as

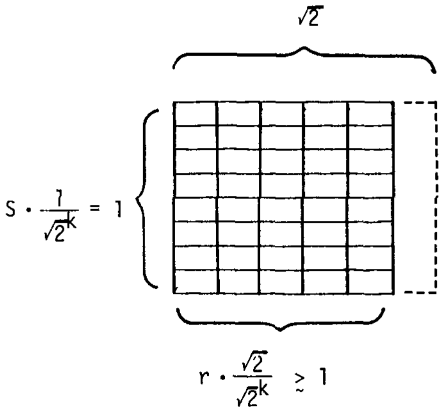
<u>Theorem 1</u>: Let $n \geq 0$ be even, and P be a set of n points in the $\sqrt{2}$ by 1 rectangle. Then rcost(P) $\leq (1 + \frac{1}{\sqrt{2}})\sqrt{n} + \sqrt{3} - \sqrt{6} + O(\frac{1}{\sqrt{n}}) \approx 1.707\sqrt{n} - .717 + O(\frac{1}{\sqrt{n}})$. Furthermore, this bound is asymptotically achievable (in particular, when $n = b_k$ for some $k \geq 0$).

So far we have considered the performance of the rectangle algorithm on points in the $\sqrt{2}$ by 1 rectangle. However, the fixed region matching problem is usually considered on the 1 by 1 square. Therefore we now adapt the rectangle algorithm to the unit square as follows. Given a set of n points P in the unit square (i.e. for all $(x, y) \in P$, $0 \leq x \leq 1$, $0 \leq y \leq 1$), we perform the rectangle algorithm treating P as a set in the rectangle defined by $[0, \sqrt{2}] \times [0, 1]$, as shown below:



The unit square is shown in solid line; the $\sqrt{2}$ by 1 rectangle is in dotted. We now upper bound rcost(P).

For the analysis, choose some even integer $k \geq 0$. Let r be the least integer such that $r \cdot \frac{\sqrt{2}}{\sqrt{2}^k} \geq 1$. Let $s = \frac{1}{\frac{1}{\sqrt{2}^k}} = \sqrt{2}^k$. Note that each level k rectangle has vertical length $\frac{1}{\sqrt{2}^k}$ and horizontal length $\frac{\sqrt{2}}{\sqrt{2}^k}$, since k is even (the proof is a simple induction on k). Therefore the unit square, and hence P, lies within the leftmost set of r·s level k rectangles:



Let $d = r \cdot \frac{\sqrt{2}}{\sqrt{2}^k}$. Our strategy is to upper bound the cost of the rectangle algorithm on an arbitrary set in the d by 1 rectangle. Since $d \geq 1$, this bound will also upper bound rcost(P).

So let Q be a set of points in the d by 1 region, $n = |Q|$. Let $\text{rcost}_k(Q) = \text{rcost}(Q) - \sum_{i=0}^{k-1}$ (sum of lengths of all edges produced at the $i^{\underline{th}}$ level of recursion by the algorithm on Q). Since there are 2^i level i rectangles, and since the length of an edge produced at the $i^{\underline{th}}$ level is at most $\frac{\sqrt{3}}{\sqrt{2}^i}$, we have that $\text{rcost}_k(Q) \geq \text{rcost}(Q)$ $- \sum_{i=0}^{k-1} 2^i \cdot \frac{\sqrt{3}}{\sqrt{2}^i} = \text{rcost}(Q) - O(\sqrt{2}^k)$.

$\therefore$ $\text{rcost}(Q) \leq \text{rcost}_k(Q) + O(\sqrt{2}^k)$. We now upper bound $\text{rcost}_k(Q)$, which is the sum of the lengths of the edges produced at levels $\geq k$. There are rs level k rectangles which compose the d by 1 region containing Q. Call these rectangles R_j, $1 \leq j \leq rs$. Let $t = rs$. For all $1 \leq j \leq t$, let $n_j = |R_j(Q)|$. By theorem 1, for all $1 \leq j \leq t$, the sum of the lengths of the edges produced within R_j is $\leq \frac{1}{\sqrt{2}^k} \cdot C_{n_j} = \frac{1}{\sqrt{2}^k}[(1 + \frac{1}{\sqrt{2}})\sqrt{n_j} + \sqrt{3}$ $- \sqrt{6} + O(\frac{1}{\sqrt{n_j}})]$. (The factor $\frac{1}{\sqrt{2}^k}$ is to scale the cost down to level k).

$$\therefore\ \text{rcost}_k(Q) \leq \frac{1}{\sqrt{2}^k}\sum_{j=1}^{t}[(1 + \tfrac{1}{\sqrt{2}})\sqrt{n_j} + \sqrt{3} - \sqrt{6} + O(\tfrac{1}{\sqrt{n_j}})]$$

$$= \frac{1}{\sqrt{2}^k}(1 + \tfrac{1}{\sqrt{2}})\sum_{j=1}^{t}\sqrt{n_j} + O(t)$$

$$= \frac{1}{\sqrt{2}^k}(1 + \tfrac{1}{\sqrt{2}^k})(\sum_{j=1}^{t-1}\sqrt{n_j} - \sqrt{n - \sum_{j=1}^{t-1} n_j}) + O(t).$$

Define the function $f : \mathbb{R}^{t-1} \to \mathbb{R}$ by

$$f(x_1, x_2, \ldots, x_{t-1}) = \sum_{j=1}^{t-1}\sqrt{x_j} - \sqrt{n - \sum_{j=1}^{t-1} x_j}.$$

Taking partial derivatives shows that f is maximized at $x_1 = x_2 = \ldots = x_{t-1} = n - \sum_{j=1}^{t-1} x_j = \frac{n}{t}$.

$$\therefore\ \text{rcost}_k(Q) \leq \frac{1}{\sqrt{2}^k}(1 + \tfrac{1}{\sqrt{2}})t\sqrt{\tfrac{n}{t}} + O(t)$$

$$= \frac{1}{\sqrt{2}^k}(1 + \tfrac{1}{\sqrt{2}})\sqrt{rs}\,\sqrt{n} + O(rs)$$

$$= \frac{1}{\sqrt{2}^k}(1 + \tfrac{1}{\sqrt{2}})\sqrt{\frac{d\sqrt{2}^k}{\sqrt{2}} \cdot \sqrt{2}^k}\,\sqrt{n} + O(2^k)$$

$$= \frac{\sqrt{d}}{\sqrt{\sqrt{2}}}(1 + \tfrac{1}{\sqrt{2}})\sqrt{n} + O(2^k).$$

$$\therefore\ \text{rcost}(Q) \leq \text{rcost}_k(Q) + O(\sqrt{2}^k) = \frac{\sqrt{d}}{\sqrt{\sqrt{2}}}\sqrt{n} + O(2^k).$$

By the definition of d, we have that $d \to 1$ as $k \to \infty$. Thus, for all $\varepsilon > 0$, we have

$$\text{rcost}(Q) \leq (1 + \varepsilon)\frac{1}{\sqrt{\sqrt{2}}}(1 + \tfrac{1}{\sqrt{2}})\sqrt{n} + O(1)$$

$$\approx (1 + \varepsilon)1.436\sqrt{n} + O(1).$$

For example, we can take $k = 10$ and hence $r = 23$, $s = 32$, $d = \frac{23}{32}\sqrt{2} \approx 1.016$, and therefore

$$\text{rcost}(Q) \leq \sqrt{\frac{23}{32}}(1 + \tfrac{1}{\sqrt{2}})\sqrt{n} + O(1)$$

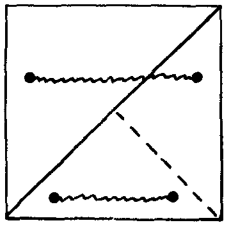
$$\approx 1.447\sqrt{n} + O(1).$$

In order to show the tightness of this bound, we again choose some even $k \geq 0$, but this time let r be the greatest integer such that $r \cdot \frac{\sqrt{2}}{\sqrt{2}^k} \leq 1$. Then let $d \cdot \frac{\sqrt{2}}{\sqrt{2}^k} \geq 1$, and $s = \sqrt{2}^k$ as before. Construct a set Q' in the d by 1 region, so that each of the rs level k rectangles in that region contains a balanced $\frac{n}{rs}$ point set. We choose $n = |Q'|$ so that $\frac{n}{rs} = b_i$ for some i, thus making $C_{\frac{n}{rs}}$ asymptotic to $(1 + \frac{1}{\sqrt{2}})\sqrt{\frac{n}{rs}}$. A similar analysis to the above shows that $\text{rcost}(Q') \geq$ $\frac{\sqrt{d}}{\sqrt{\sqrt{2}}}(1 + \frac{1}{\sqrt{2}})\sqrt{n} - O(2^k)$. Hence $(\forall\varepsilon > 0)(\exists$ set Q' in the unit square)$[\text{rcost}(Q') \geq (1 - \varepsilon)\frac{1}{\sqrt{\sqrt{2}}} \cdot$ $(1 + \frac{1}{\sqrt{2}})\sqrt{n} - O(1) \approx (1 - \varepsilon)\ 1.436\sqrt{n} - O(1)]$.

The reader may wonder why we did not simply choose some k such that the 1 by 1 square can be exactly tessellated by level k rectangles (i.e. we would have $d = 1$). Unfortunately, as is easily shown, no such k exists.

In summary, $1.436 \approx \frac{1}{\sqrt{\sqrt{2}}}(1 + \frac{1}{\sqrt{2}})$ $= \inf\{x$: for all n-point sets P in the unit square, $\text{rcost}(P) \leq x\sqrt{n} + O(\sqrt{n})\}$, where inf denotes the greatest lower bound.

A square can be partitioned into two 45° - 45° - 90° triangles. Also, a 45° - 45° - 90° triangle can be partitioned into two 45° - 45° - 90° subtriangles of equal size. This suggests a second partition algorithm, which we call the triangle algorithm: given a set P of n points in the unit square, do exactly as the rectangle algorithm, except that when a region is split, it is split into two equal sized 45° - 45° - 90° triangles. An example in which n = 4 is shown below.



Here the first split is along the main diagonal; the second split is shown in dotted line. The matching produced is in jagged line.

In analogy to the previous section, define a triangle to be either (i) one of the two main 45° - 45° - 90° triangles with hypoteneuse $\sqrt{2}$, into which the square is split, or (ii) one of the two 45° - 45° - 90° subtriangles into which a triangle may be split. Furthermore, if T is a triangle, then let

$$\text{level}(T) = \begin{cases} 0, & \text{if } T \text{ is a main triangle (of hypoteneuse length } \sqrt{2}) \\ \text{level}(T') + 1, & \text{otherwise, where } T' \text{ is the triangle which splits into } T \text{ and some other triangle.} \end{cases}$$

Note that the level of a triangle is 1 less than the level of recursion on which the triangle lies (in contrast to the level of a rectangle in the previous section, which equals the level of recursion on which it lies). We define level in this way because our strategy is to analyze the worst case cost of points in a main triangle, and then use that result to analyze the worst case cost for points in the unit square.

If P is a set of points in the unit square, then let tcost(P) = the sum of the lengths of the edges in the matching produced by the triangle algorithm on P. For all $n \geq 0$, let E_n = sup{tcost(P): P is a set of n points in a main triangle of the unit square}, and let F_n = sup{tcost(P): P is a set of n points in the unit square}. As mentioned above, we will first analyze the E_n and then use that result to analyze the F_n.

First note that we can restrict the levels of recursion to at most $\lceil \lg n \rceil$ and so enable the algorithm to run in time $O(n \log n)$, as for the rectangle algorithm. This restriction does not affect the worst case cost, as can be proved by an argument just like lemma 1.

From here to the end of the analysis of the E_n, let "set of points" denote a set of points in a main triangle (i.e. of hypoteneuse length $\sqrt{2}$). If T is a triangle, P a set of points, then let T(P) denote the set of points of P contained in T. Define the property balanced exactly as in rectangle algorithm's analysis, except substituting the word "triangle" for "rectangle", and understanding the "opposite corners" of a triangle to mean its two 45° corners. In analogy to the rectangle results, we now show the balanced sets to be the worst case for the triangle algorithm.

Lemma 2': Let $n \geq 0$ be even, and P a set of n points. Then ($\exists$ set of points Q)[$|Q|$ = n and tcost(Q) $\geq$ tcsot(P) and ($\forall$ triangle T such that $T(Q) \geq 1$)

[1. $|T(Q)|$ even $\Rightarrow$ T splits into T_1, T_2 such that $|T_1(Q)|$, $|T_2(Q)|$ are odd, and T_1 and T_2 each strand a point of Q in a 45° corner of T,

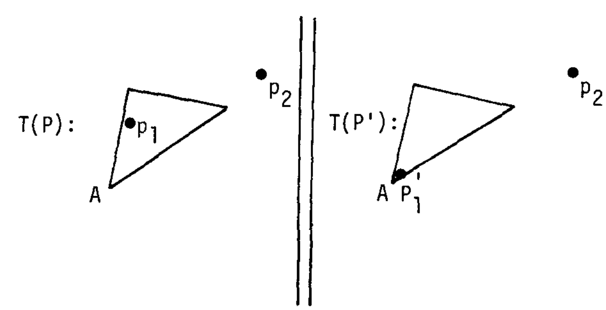
2. $|T(Q)|$ odd $\Rightarrow$ T strands a point of Q in one of its own 45° corners,

3. $|T(Q)| \geq 2 \Rightarrow$ the two subtriangles of T each contain at least 1 point of Q]].

Proof: Another rearranging argument, very similar to that of lemma 2. Say a triangle T is even if $|T(P)|$ is even otherwise T is odd. The rearranging argument for triangles is slightly more complicated than that for rectangles, since the farthest part of an odd triangle from some point may be a 90° corner rather than a 45° corner. To handle this situation, we make use of the following terms: if a triangle T splits into subtriangles T_1 and T_2, then we say that T is the father of T_1 and T_2, and that T_1 and T_2 are brothers.

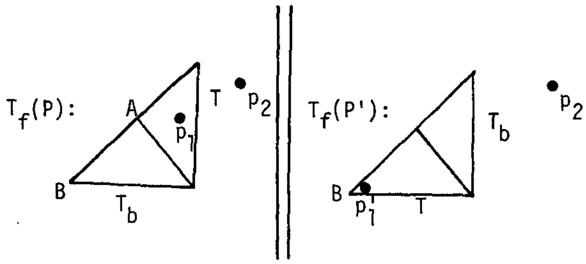
First we rearrange all triangles T such that $|T(P)| = 1$. Let T be such a triangle, and p_1 the point in T. Let T_b be the brother of T, and T_f the father (T_b and T_f must exist since n is even and T is odd). Let p_2 be the point matched to p_1 by the algorithm. Let A denote the corner of T which is farthest from p_2.

Case 1: A is a 45° corner of T. Then simply "move" p_1 to A, without decreasing the cost:



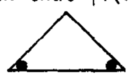
(We will not explicitly define P' in this proof as we did in the proof of lemma 2. It should be clear by now how we "move" points.)

Case 2: A is a 90° corner of T. Then the farthest part of T_f from p_2 must be some 45° corner B of T_f.



Thus move p_1 to a 45° corner of T and then swap T with T_b. This does not affect any other matches since T_b is even.

For triangles T such that $|T(P)| = 2$, merely note that the arrangement



gives the greatest cost.

Now assume we have rearranged all triangles T such that $|T(P)| \leq K$ for some $K \geq 2$. Let T be a triangle such that $|T(P)| = K + 1$. Let T_1, T_2 be the subtriangles of T, T_b the brother of T, and T_f the father of T.

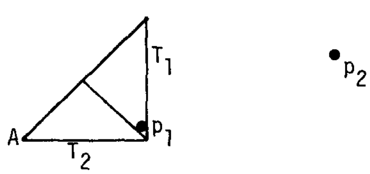
Case 1: K + 1 is odd. Assume WLOG that T_1 is odd, T_2 even.

Case 1.1: $|T_2(P)| = 0$. Handle this just as in the proof of Lemma 2; namely, move 2 points out of the corners of T_1's even subtriangle into T_2's corers.

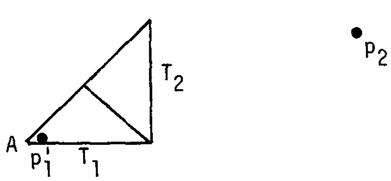
Case 1.2: $|T_2(P)| > 0$. T strands some point p_1, which is matched to some point p_2 outside of T. Let A be the corner of T which is farthest from p_2. Since $|T_1(P)| \leq K$, p_1 is already in a 45° corner of T_1.

Case 1.2.1: A is a 45° corner of T. Then if p_1 is not already in A, then rotate T_1 and then swap T_1 with T_2 (if necessary) to put p_1 in A, e.g.

T(P):

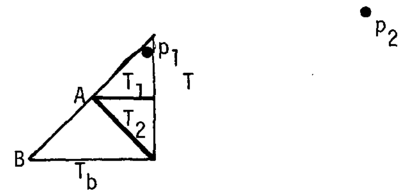


T(P'):



Case 1.2.2: A is a 90° corner of T. Then the farthest part of T_f from p_2 is some 45° corner B of T_f.

$T_f(P)$:



Note that B is also a 45° corner of T_b. Therefore swap T_b with T and rearrange T using Case 1.2.1. (This affects no matchings of points other than p_1 and p_2, since T_b is even. We know that T_b is even since $p_2 \notin T_b(P)$, which we know since the farthest corner from any point in T_b must be a 45° corner of T).

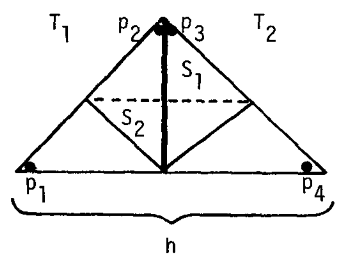
Case 2: K + 1 is even. Assume WLOG that $|T_1(P)| \geq |T_2(P)|$.

Case 2.1: $|T_2(P)| = 0$. Then proceed as in Case 1.1.

Case 2.2: $|T_2(P)| > 0$. Then both T_1 and T_2 both have already been rearranged.

Case 2.2.1: $|T_1(P)|$, $|T_2(P)|$ both even. Thus,

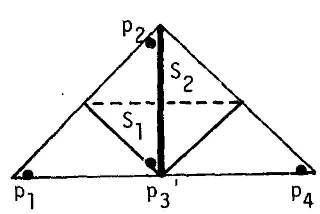
T(P):



That is, T is a triangle of hypoteneuse length h for some h > 0. T_1 matches points p_1 and p_2 in its opposite corners. T_2 matches points p_3 and p_4 in its opposite corners. S_2 is the even subtriangle of T_1 which strands p_2. S_1 is the odd subtriangle of the subtriangle of T_2 which strands p_3.

Now let P' be like P except that the points in S_1 have been swapped with those in S_2:

T(P'):



Hence $tcost(P) = d(p_1, p_2) + d(p_3, p_4) + c$ for some $c > 0$, and $tcost(P') = d(p_1, p_4) + d(p_2, p_3') + c$.

Now $d(p_1, p_2) = d(p_3, p_4) = \frac{h}{\sqrt{2}}$, $d(p_1, p_4) = h$, $d(p_2, p_3') = \frac{h}{2}$. $\therefore$ $tcost(P) = h\sqrt{2} + c < \frac{3}{2}h + c = tcost(P')$, as desired; so let $P = P'$, and continue.

<u>Case 2.2.2</u>: $|T_1(P)|, |T_2(P)|$ both odd. Then T_1 strands a point p_1 in one of its 45° corners, and T_2 strands a point p_2 in one of its 45° corners. If p_1 and p_2 are not both in 45° corners of T, then rotate T_1 or T_2 or both to put them there.

Finally, let Q be this rearranged version of P. Q satisfies the properties stated in the Lemma. <u>QED Lemma 2'</u>

<u>Lemma 3'</u>: Let $n \geq 0$ be even, P a set of n points. Then $(\exists$ set of points $Q)[|Q| = n$ and $tcost(Q) \geq tcost(P)$ and Q is balanced]. The proof is identical to that for Lemma 3, substituting "triangle" for "rectangle".

Thus $(\forall$ even $n \geq 0)[E_n = tcost(P)$, where P is a balanced n point set]. The length of a level i hypoteneuse is $\frac{\sqrt{2}}{\sqrt{2}^i} = \frac{\sqrt{2}}{\sqrt{3}}\left(\frac{\sqrt{3}}{\sqrt{2}^i}\right) = \frac{\sqrt{2}}{\sqrt{3}}$(length of a diagonal in a level i rectangle). $\therefore$ for all even $n \geq 0$, $E_n = \frac{\sqrt{2}}{\sqrt{3}} C_n \leq \frac{\sqrt{2}}{\sqrt{3}}[(1 + \frac{1}{\sqrt{2}})\sqrt{n} + \sqrt{3} - \sqrt{6} + O(\frac{1}{\sqrt{n}})]$. Note that for all odd $n \geq 0$, $E_n \leq E_{n-1}$. To see this, let P be a set of points, $|P| = n$ be odd. Then there is some $p_1 \in P$ such that p_1 is not matched to any other point by the algorithm. Then $tcost(P) = tcost(P - \{p_1\})$, and hence $E_n \leq E_{n-1}$. $\therefore$ for all $n \geq 0$, $E_n \leq \frac{\sqrt{2}}{\sqrt{3}}(1 + \frac{1}{\sqrt{2}})\sqrt{n} + O(1)$.

Now we analyze the F_n, which are our primary interest. Let P be a set of points in the unit square. The square is split into two main triangles, one with m points and one with n-m points, for some $0 \leq m \leq n$. $\therefore$ $tcost(P) \leq \max_{0 \leq m \leq n}\{E_m + E_{n-m}\} + \sqrt{2} \leq \max_{0 \leq m \leq n}\{\frac{\sqrt{2}}{\sqrt{3}}(1 + \frac{1}{\sqrt{2}}) \cdot (\sqrt{m} + \sqrt{n-m})\} + O(1)$.

Treating $\sqrt{m} + \sqrt{n-m}$ as a real function of m and differentiating shows that $\sqrt{m} + \sqrt{n-m}$ is maximized at $m = \frac{n}{2}$. $\therefore$ $tcost(P) \leq \frac{\sqrt{2}}{\sqrt{3}}(1 + \frac{1}{\sqrt{2}})\, 2\sqrt{\frac{n}{2}} + O(1) = \frac{2}{\sqrt{3}}(1 + \frac{1}{\sqrt{2}})\sqrt{n} + O(1)$. Thus for all $n \geq 0$, $F_n \leq \frac{2}{\sqrt{3}}(1 + \frac{1}{\sqrt{2}})\sqrt{n} + O(1) \simeq 1.97\sqrt{n} + O(1)$. This bound is asymptotically achievable, since if $n = 2b_r$ for some $r \geq 0$, then we can construct a set P such that the unit square splits into T_1, T_2 such that $T_1(P)$ and $T_2(P)$ are both balanced b_r point sets. $\therefore$ as shown in the previous section, since $\frac{C_{b_r}}{\sqrt{b_r}} \to (1 + \frac{1}{\sqrt{2}})$ as $r \to \infty$, we have that

$$\frac{tcost(P)}{\sqrt{n}} \to \frac{2}{\sqrt{3}}(1 + \frac{1}{\sqrt{2}}) \text{ as } n \to \infty.$$

Our third partitioning method, the <u>Square-Rectangle Algorithm</u>, works just like the rectangle or triangle heuristics, except that the regions are partitioned as follows. We start off with n points in the unit square. The square is split vertically into two 1 by $\frac{1}{2}$ rectangles. These rectangles are then each split into two $\frac{1}{2}$ by $\frac{1}{2}$ squares. (As in the last two algorithms, we do this splitting only if the region has ≥ 2 points in it and is at or below the $\lceil \lg n \rceil^{\text{th}}$ level of recursion, counting the unit square as level 0.) In general, each square is split vertically into two rectangles of ratio 2 to 1 between the vertical and horizontal sides; and each rectangle is split into two squares.

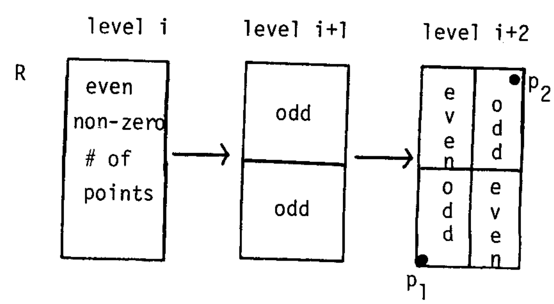
We do not yet know how to put a tight upper bound on the cost of the matching produced by this algorithm. A very crude upper bound can be derived by assuming that each region (square or rectangular) matches two points in its opposite corners, thus

$$\text{cost} \leq \sum_{\substack{0 \leq i \leq \lg n+1 \\ i \text{ even}}} 2^i \frac{\sqrt{2}}{\sqrt{2}^i} + \sum_{\substack{0 \leq i \leq \lg n+1 \\ i \text{ odd}}} 2^i \cdot \frac{\sqrt{5}}{\sqrt{2}^{i+1}}$$

$$\leq (\sqrt{20} + \sqrt{8})\sqrt{n} + O(1) \approx 7.30\sqrt{n} + O(1).$$

Certainly the least upper bound is much lower than this; we merely wanted to show the cost to be bounded by $O(\sqrt{n})$. Below we construct an example in which the cost is asymptotic to $\frac{3}{2}\sqrt{n}$.

Let P be a set of points in the unit square such that each even square splits into two even rectangles, and each even rectangle R splits into odd squares S_1, S_2 such that S_1 and S_2 strands points in opposite corners of R. A region is <u>even</u> if it contains an even number of points of P, otherwise it is <u>odd</u>. Assume P is full to some level 2r+1 in the sense that each level 2(r-1) + 1 rectangle has exactly 1 or 2 points in it. We can so construct P using the technique used to construct full sets for the rectangle algorithm (see above). Thus if R is a rectangle of level i for some $1 \leq i \leq 2(r-1) + 1$, then



Note that if a level i rectangle R is odd then R splits into three even and one odd level i + 2 rectangles. If R is even then it splits into two even and two odd level i + 2 rectangles. For all $0 \leq i \leq r-1$, let E_i = the number of even rectangles of level 2i+1, O_i = the number of odd rectangles of level 2i+1. (Note that a level K consists of rectangles $\Leftrightarrow$ K is odd). Then

$$E_0 = 2,\ O_0 = 0.$$

$$\forall 1 \leq i \leq r-1,\ E_i = 2E_{i-1} + 3\,O_{i-1},$$

$$O_i = 2E_{i-1} + O_{i-1}.$$

$$E_i + O_i = 2 \cdot 4^i.$$

The solution to these equations is $E_i = \frac{6}{5}4^i + \frac{4}{5}(-1)^i$, $O_i = \frac{4}{5}4^i - \frac{4}{5}(-1)^i$, $\forall\ 0 \leq i \leq r-1$.

Let $n = |P|$. Then

$$n = \sum_{i=0}^{r-1} 2 \cdot E_i = \frac{4}{5}4^r + \frac{4}{5}(-1)^{r-1},$$ and hence

$r = \log_4(\frac{5}{4}n) \pm O(1)$. Since the length of a level 2i+1 diagonal is $\frac{\sqrt{5}}{2^{i+1}}$, we have

$$\text{cost}(P) = \sum_{i=0}^{r} E_i \cdot \frac{\sqrt{5}}{2^{i+1}} \geq \frac{3}{2}\sqrt{n} - O(1).$$

We conjecture that the asymptotic worst case cost for this algorithm is very close to $\frac{3}{2}\sqrt{n}$.

The last partitioning method we consider, the Four-Square Algorithm, works as follows. Each square S (initially the unit square) which has ≥ 2 input points in it is split into 4 equal subsquares. The algorithm is applied recursively to each of these subsquares. Then the best matching of the ≤ 4 stranded points is made (the best matching of 3 points is the closest pair). In analogy to the other partitioning algorithms, if a square S contain ≥ 2 points and is on the $(\lceil\log_4 n\rceil + 1)^{\underline{rst}}$ level of recursion, then arbitrarily match up the points in S until 0 or 1 is left. Thus this algorithm also runs in time $O(n \log n)$.

As for the square-rectangle heuristic, we have no tight upper bound for this algorithm, but know it to be $\Theta(\sqrt{n})$. As for a lower bound, we construct an example below of cost $\frac{\sqrt{2}}{\sqrt{3}}(1 + \frac{1}{\sqrt{2}})\sqrt{n} - O(1) \approx 1.39\sqrt{n} - O(1)$.

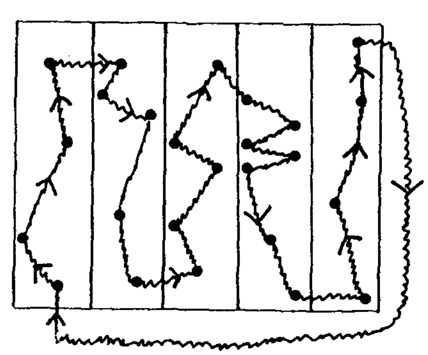
Construct a set P of points in the unit square such that each even square S splits into S_1, S_2, S_3, S_4 such that S_1, S_3 odd, S_2, S_4 even, and S_1 and S_3 strand points in opposite corners of S. Also, each odd square S splits into odd squares S_1, S_2, S_3, and even square S_4 such that each of the 3 points stranded by S_1, S_2, S_3 is in a different corner of S. Thus at level i, each even square contributes an edge of length $\frac{\sqrt{2}}{2^i}$, and each odd square contributes an edge of length $\frac{1}{2^i}$. Make P such that for some integer r, each level r-1 square has either 1 or 2 points of P in it. For all $0 \leq i \leq r-1$, let E_i = the number of even level i squares and O_i = the number of odd level i squares. Then $E_0 = 1$, $O_0 = 0$, and $(\forall 1 \leq i \leq r-1)$ $[E_i = 2E_{i-1} + O_{i-1},\ O_i = 2E_{i-1} + 3O_{i-1},\ E_i + O_i = 4^i]$. The solution is $E_i = \frac{1}{3}4^i + \frac{2}{3}$, $O_i = \frac{2}{3}4^i - \frac{2}{3}$. Let $n = |P|$. Note that $n = O_{r-1} + 2E_{r-1}$ (since each level r-2 square has 5 or 6 points, each level r-1 square has 1 or 2 points, and each level r+1 square has 0 or 1 point). $\therefore$ $n = \frac{1}{3}4^r - \frac{2}{3}$, and hence $r = \log_4(3n + 2)$. $\therefore$ $\text{cost}(P) = \sum_{i=0}^{r-1} E_i \cdot \frac{\sqrt{2}}{2^i} + \sum_{i=0}^{r-2} O_i \cdot \frac{1}{2^i} = \frac{\sqrt{2}}{\sqrt{3}}(1 + \frac{1}{\sqrt{2}})\sqrt{n} + \sqrt{2} - 2 + O(\frac{1}{\sqrt{n}}) \approx 1.39\sqrt{n} - O(1)$. Incidentally, this expression is exactly the same as that found to upper bound the cost of the triangle algorithm on n points in a main triangle. We have no geometric explanation for this coincidence.

Comparing these results (see summary) we conclude that the best (in terms of worst case performance) partition method is either the rectangle or the four-square. If indeed the four-square is superior, then the rectangle is a close second.

The Strip Algorithm

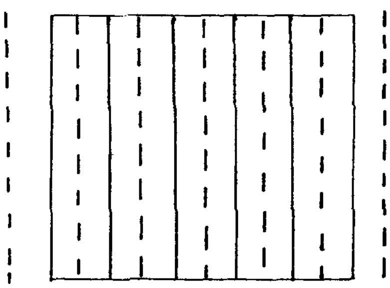
This algorithm is a modification of one analyzed for expected performance in Papadimitriou [10].

Let $r = \lceil\frac{\sqrt{n}}{\sqrt{2}}\rceil$. The unit square is divided into r vertical strips, each of width $\frac{1}{r}$. Then a traveling salesman tour T_1 is constructed by starting at the lowest input point in the leftmost strip, going up that strip in the path which includes all input points of that strip, then down the next strip, up the next, etc., and finally returning to the starting point, as shown:



Here T_1 is shown in jagged line. For ease of drawing, not all of the input points are pictured here (since in order to have $r = 5$ strips there must be $50 \leq n < 72$ input points).

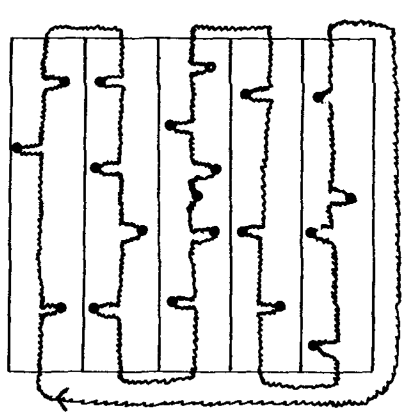
Then, a second traveling salesman tour T_2 is constructed in the same way, except that here the strip boundaries have been shifted by $\frac{1}{2} \cdot \frac{1}{r}$ to the right. The strip boundaries for T_1 are shown below as solid lines, those for T_2 in dashed:



Thus there are $r + 1$ strips used in constructing T_2, each of width $\frac{1}{r}$. Note that the leftmost of these strips contains no input points in its left half. Similarly the rightmost strip contains no input points in its right half.

Thus we have two traveling salesman tours T_1 and T_2. Since n is even, each tour contains exactly two matchings. The algorithm outputs the shortest of these four matchings.

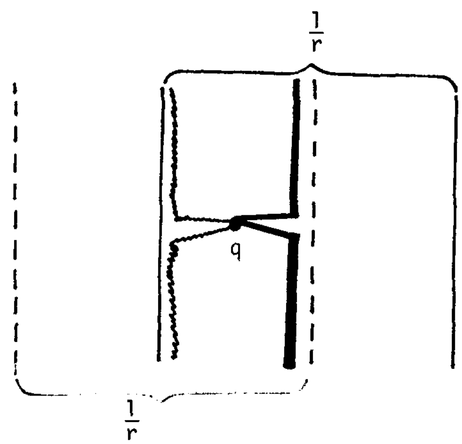
To upper bound the cost of the matching produced, consider paths P_1 and P_2 defined as follows: P_1 starts at the bottom, on the median of the leftmost of the strips used in constructing T_1. P_1 follows the median of the strip up to the top, then down the median of the next strip, up the next, etc. For each strip, for each point in that strip, the path P_1 juts out to that point and then back to the median, moving at right angles, as illustrated: (P_1 is in jagged)



The path P_2 is defined like P_1, except that P_2 follows the medians of the strips used to construct T_2.

It follows from the triangle inequality that length $(T_1) \leq$ length (P_1) and length $(T_2) \leq$ length (P_2). Our strategy is to upper bound length (P_1) + length (P_2).

Consider some input point q. q must lie in some strip (shown below between solid lines) used for T_1 and P_1, and in some strip (between dashed lines) used for T_2 and P_2:



A segment of P_1 is shown in heavy solid line, and a segment of P_2 in jagged line. It should be clear that the total amount of horizontal line of P_1 or P_2 which juts out to q and back is $2(\frac{1}{2} \cdot \frac{1}{r}) = \frac{1}{r}$. Since q was arbitrary, there is a total of $\frac{n}{r}$ units of horizontal line in P_1 and P_2 together which jut out to points and back. Also, P_1 has $r \cdot 1 = r$ units of vertical line (i.e., r strips of length 1). P_2 has $r + 1$ strips and hence $r + 1$ units of vertical line. P_1 has $1 - \frac{1}{r}$ units of horizontal line which run from the end of one strip to the start of the next. P_2 has 1 unit of such line. Finally, P_1 and P_2 each have a segment of length less than $\sqrt{2}$ which joins the end of the last strip back to the starting position.

Thus, in total, length (T_1) + length(T_2)

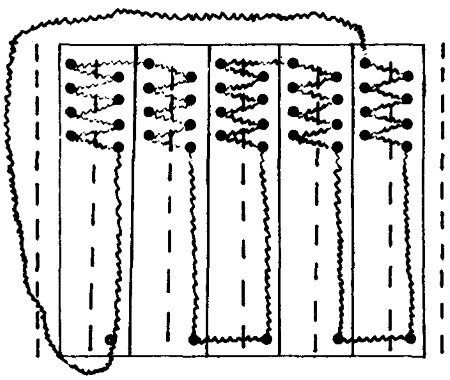
$$\begin{aligned}
&\leq \frac{n}{r} + r + (r+1) + (1-\frac{1}{r}) + 1 + \sqrt{2} + \sqrt{2} \\
&< \frac{n}{r} + 2r + 3 + 2\sqrt{2} \\
&= \frac{n}{\lceil \frac{\sqrt{n}}{\sqrt{2}} \rceil} + 2\lceil \frac{\sqrt{n}}{\sqrt{2}} \rceil + 3 + 2\sqrt{2} \\
&\leq 2\sqrt{2}\sqrt{n} + 5 + 2\sqrt{2} \\
&= 2\sqrt{2}\sqrt{n} + O(1).
\end{aligned}$$

Thus, $\min\{\text{length}(T_1), \text{length}(T_2)\}$
$\leq \sqrt{2}\sqrt{n} + O(1)$.

Therefore the cost of the matching produced is $\leq \frac{1}{2}\min\{\text{length}(T_1), \text{length}(T_2)\}$

$$\leq \frac{1}{\sqrt{2}}\sqrt{n} + O(1) \approx .707\sqrt{n} + O(1).$$

This bound is asymptotically achievable, as shown by the following example:



T_1 is shown in jagged line. T_2 is not shown, but looks almost like T_1 shifted by $\frac{1}{2r}$ to the right. The points are arranged so that halfway between each solid vertical line and either of its two neighboring dotted vertical lines, there is a vertical string of $O(\sqrt{n})$ points. Intuitively, these points are placed so that T_1 and T_2 must zigzig and hence look very much like P_1 and P_2, respectively. This attains the maximum amount (neglecting lower order terms) of horizontal line for T_1 and T_2. There is a point at the bottom of each strip, so as to attain the maximum vertical length. A simple computation shows length(T_1), length(T_2) $\approx \sqrt{2}\sqrt{n} + O(1)$, and also that the cost of the matching is $\frac{1}{\sqrt{2}}\sqrt{n} + O(1)$.

The algorithm can be implemented in time $O(n \log n)$ using sorting. Note that the strip algorithm can be used to obtain a traveling salesman tour (i.e., the shorter of $\{T_1, T_2\}$) in the unit square, of length at most $\sqrt{2}\sqrt{n} + O(1)$. These results generalize easily to a 1 by x region giving a matching whose cost is at most $\sqrt{\frac{x}{2}}\sqrt{n} + O(1)$ and a traveling salesman tour whose cost is at most $\sqrt{2x}\,\sqrt{n} + O(1)$.

Decomposition Algorithm

This last matching algorithm is a hybrid between Edmond's $O(n^3)$ time optimizing algorithm, and any of the $O(n \log n)$ time heuristics. The resulting algorithm has the best properties of both: an $O(n \log n)$ time bound and a cost bound which is the same, neglecting lower order terms, as that for the optimizing. In the following presentation of the algorithm, we happen to choose the strip heuristic as our $O(n \log n)$ heuristic:

1. $c \leftarrow \left\lceil \frac{\sqrt{n}}{\sqrt{\sqrt{\lg n}}} \right\rceil$
2. Partition the unit square into c^2 subsquares of equal size.
3. For each of these subsquares, perform the optimizing algorithm iteratively on sets of K input points chosen arbitrarily from that subsquare, where K is the largest even integer $\leq \min\{4 \cdot \left\lceil \frac{n}{c^2} \right\rceil$, number of input points still unmatched in the subsquare$\}$, until the subsquare is left with 0 or 1 point in it.
4. Perform the strip heuristic on the remaining $< c^2$ points.
5. Output the union of the matchings found in steps 3 and 4, and halt.

In order to analyze the algorithm's performance, let

$\alpha = \inf\{x\colon x\sqrt{n} + o(\sqrt{n})$ upper bounds the worst case cost of the optimizing algorithm$\}$.

We know that α exists and that $.537 \approx \frac{1}{\sqrt{\sqrt{12}}} \leq \alpha \leq \frac{1}{\sqrt{2}} \approx .707$, since $\frac{1}{\sqrt{\sqrt{12}}}\sqrt{n} + O(1)$ is the cost of the optimal matching of n points on a 1 by 1 hexagonal grid, and since $\frac{1}{\sqrt{2}}\sqrt{n} + O(1)$ is the upper bound for the strip algorithm. (We suspect that α is close to $\frac{1}{\sqrt{\sqrt{12}}}$, but have been unable to prove it). We will show that the decomposition algorithm produces a matching of cost $\leq \alpha\sqrt{n} + o(\sqrt{n})$. Thus, in an asymptotic sense, the decomposition algorithm's performance is as good as possible.

Let $b = \left\lceil \frac{n}{c^2} \right\rceil$. Number the subsquares from 1 to c^2. For all $1 \leq i \leq c^2$, let B_i denote the set of input points originally in the $i^{\underline{th}}$ subsquare and let $b_i = |B_i| \bmod 4b$. Thus $\frac{|B_i| - b_i}{4b} + 1 \geq$ the number of calls to the optimizing algorithm on the $i^{\underline{th}}$ subsquare. Finally, let $t = \sum_{i=1}^{c^2} \frac{|B_i| - b_i}{4b}$. Thus $t + c^2 \geq$ the total number of calls to the optimizing algorithm. Note that $t = \frac{1}{4b}\left(n - \sum_{i=1}^{c^2} b_i\right)$.

Now for all $r \geq 1$, the cost of the matching produced by the optimizing algorithm on r points in a $\frac{1}{c}$ by $\frac{1}{c}$ square is at most $\frac{1}{c}(\alpha\sqrt{r} + o(\sqrt{r}))$. The $\frac{1}{c}$ factor scales down the cost from the unit square to the $\frac{1}{c}$ by $\frac{1}{c}$ square. Thus the sum of the costs of all calls to the optimizing algorithm is at most

$$\frac{1}{c}(t(\alpha\sqrt{4b} + o(\sqrt{b})) + \sum_{i=1}^{c^2} (\alpha\sqrt{b_i} + o(\sqrt{b_i})))$$

$$= \frac{1}{c}(t\alpha\sqrt{4b} + \alpha \sum_{i=1}^{c^2} \sqrt{b_i} + o(c^2\sqrt{b})), \text{ since}$$

$b_i \leq 4b$ for all i, $1 \leq i \leq c^2$, and since $t \leq c^2$.

The matching produced in step 4 by the strip algorithm on at most c^2 points in the unit square has cost $\leq \frac{1}{\sqrt{2}}\sqrt{n} + O(1)$. Therefore, the total cost of the matching is at most

$$(1)\quad \frac{\alpha}{c}\left(t\sqrt{4b} + \sum_{i=1}^{c^2} \sqrt{b_i} + o(c^2\sqrt{b})\right) + \frac{c}{\sqrt{2}}$$

We now show that $t\sqrt{4b} + \sum_{i=1}^{c^2} \sqrt{b_i}$ is maximized when $b_1 = b_2 = \ldots = b_{c^2} = b$. Let $f : \mathbb{R}^{c^2} \to \mathbb{R}$ be defined by

$$f(b_1, b_2, \ldots, b_{c^2}) = t\sqrt{4b} + \sum_{i=1}^{c^2} \sqrt{b_i}$$

$$= \frac{1}{4b}\left(n - \sum_{i=1}^{c^2} b_i\right)\sqrt{4b} + \sum_{i=1}^{c^2}\sqrt{b_i}$$

$$= (n - \sum_{i=1}^{c^2} b_i) \cdot \frac{1}{\sqrt{4b}} + \sum_{i=1}^{c^2} \sqrt{b_i}. \text{ Then}$$

for all i, $1 \leq i \leq c^2$, $\frac{\partial f}{\partial b_i} = \frac{1}{\sqrt{4b}} + \frac{1}{2\sqrt{b_i}} = 0 \Leftrightarrow b = b_i$, and $\frac{\partial^2 f}{\partial^2 b_i} < 0$.

Thus f is maximized at $b_1 = b_2 = \ldots = b_{c^2} = b$.

Note that $b_1 = b_2 = \ldots = b_{c^2} = b$ implies $n \geq bc^2$, which implies $n = bc^2$ (since $b = \left\lceil \frac{n}{c^2} \right\rceil$ and hence $bc^2 \geq n$). This implies

$$t = \frac{n - \sum_{i=1}^{c^2} b_i}{4b} = \frac{n - c^2 b}{4b} = 0. \text{ Thus}$$

expression (1) is maximized when $t = 0$ and $b_1 = b_2 = \ldots = b_{c^2} = b$; hence

$$\text{cost} \leq \frac{\alpha}{c}\left(\sum_{i=1}^{c^2} \sqrt{b} + o(c^2\sqrt{b})\right) + \frac{c}{\sqrt{2}}$$

$$= \alpha c\sqrt{b} + o(c\sqrt{b}) + \frac{c}{\sqrt{2}}$$

Note that $\sqrt{b} = \sqrt{\left\lceil \frac{n}{c^2} \right\rceil} < \sqrt{\frac{n}{c^2} + 1} < \sqrt{\frac{n}{c^2}} + 1 = \frac{\sqrt{n}}{c} + 1.$

$$\therefore \text{cost} \leq \alpha\sqrt{n} + o(\sqrt{n}) + (\alpha + \tfrac{1}{\sqrt{2}})\, c$$

$$= \alpha\sqrt{n} + o(\sqrt{n}) + (\alpha + \tfrac{1}{\sqrt{2}}) \cdot \frac{\sqrt{n}}{\sqrt{\sqrt{\lg n}}}$$

$$= \alpha\sqrt{n} + o(\sqrt{n}),$$

as we claimed.

Next we show that the algorithm runs in time $O(n \log n)$. Step 2, the partitioning of the points, can be performed in time $O(n)$ as follows: for each input point p, we determine, by a few simple arithmetic operations, which subsquare contains p. We can do this since the subsquares form a grid. More precisely, we associate each subsquare with the grid point (x_0, y_0) at its lower left. Thus for each input point $p = (x, y)$, compute

$$x_0 \leftarrow \begin{cases} \left\lfloor \frac{x}{\frac{1}{c}} \right\rfloor \cdot \frac{1}{c} & \text{, if } x \neq 1 \\ 1 - \frac{1}{c} & \text{, if } x = 1, \end{cases}$$

$$y_0 \leftarrow \begin{cases} \left\lfloor \frac{y}{\frac{1}{c}} \right\rfloor \cdot \frac{1}{c} & \text{, if } y \neq 1 \\ 1 - \frac{1}{c} & \text{, if } y = 1. \end{cases}$$

Then put p in the list of input points found to be in the subsquare whose lower left corner is (x_0, y_0). Since there are $c^2 < n$ subsquares, the whole partitioning can be performed in time $O(n)$.

In step 3, there are at most $t + c^2$ calls on the cubic time optimizing algorithm, each call having $\leq 4b$ points. Thus the time for step 3 is

$$\leq (t + c^2)(4b)^3 = \left(\frac{n - \sum_{i=1}^{c^2} b_i}{4b} + c^2\right)(4b)^3$$

$$\leq \left(\frac{n}{4b} + c^2\right)(4b)^3 = O(nb^2 + c^2 b^3)$$

$$= O\left(n(\sqrt{\lg n})^2 + \left(\frac{\sqrt{n}}{\sqrt{\sqrt{\lg n}}}\right)^2 (\sqrt{\lg n})^3\right) = O(n \log n).$$

Step 4 requires time $O(c^2 \lg c^2)$

$$= O\left(\left(\frac{\sqrt{n}}{\sqrt{\sqrt{\lg n}}}\right)^2 \lg\left(\frac{n}{\sqrt{\lg n}}\right)\right) = O(n \log n).$$

Thus the total running time is $O(n \log n)$.

Decomposition Algorithm for TSP

A decomposition algorithm similar to the above can be used, with similar results, for the traveling salesman problem in the unit square. Recall that the strip algorithm gives a traveling salesman tour of length at most $\sqrt{2}\sqrt{n} + O(1)$. Also, the optimal tour of n points on a 1 by 1 hexagonal grid has length $\frac{2}{\sqrt{\sqrt{12}}}\sqrt{n} + O(1)$. Therefore there exists some real β such that $1.07 \approx \frac{2}{\sqrt{\sqrt{12}}} \leq \beta$

$\leq \sqrt{2} \approx 1.41$ and

$\beta = \inf \{x\colon x\sqrt{n} + o(\sqrt{n})$ upper bounds the worst case cost of the optimizing TSP algorithm in the unit square$\}$.

We will present a hybrid between an exhaustive optimizing algorithm and the strip heuristic. Analogously to our matching results, this hybrid has worst case cost bounded by $\beta\sqrt{n} + o(\sqrt{n})$, and runs in time $O(n \log n)$. This is particularly remarkable in that the Euclidean TSP is known to be NP-hard [6], [9].

Input: a set V of n points in the unit square.

Output: a traveling salesman tour of V.

Method:

0. if $n < 2^{2^{2^2}} = 65{,}536$

   then exhaustively search all n! permutations to find the shortest tour; halt.

   [This step is to ensure below that lglglglgn is defined and ≥ 1].

1. $c \leftarrow \left\lceil \frac{\sqrt{n}}{\sqrt{\mathrm{lglglglg}n}} \right\rceil$

2. partition the unit square into c^2 subsquares of equal size.

3. For each of these subsquares,
   a) exhaustively find the shortest tour of K input points chosen arbitrarily from that subsquare, where

   $K = \min \{4 \cdot \left\lceil \frac{n}{c^2} \right\rceil$, number of input points in the subsquare not yet chosen$\}$.

   Iterate this step until all input points of the subsquare have been chosen.

   b) if the subsquare originally has at least one input point in it then distinguish one of those points

4. Perform the strip heuristic to find a tour of the $\leq c^2$ distinguished points.

5. $T' \leftarrow$ the union of the edges in the tours found in steps 3 and 4. [Note that T' is a connected graph whose nodes are the set V. Also, T' contains an Eulerian circuit. Therefore one can convert T' into a tour T of V using the method in [3], so that, by the triangle inequality,

   $$\mathrm{length}(T) \leq \sum_{e \in T'} \mathrm{length}(e)].$$

   Output T constructed in this way, and halt.

The analysis of the worst-case cost is identical to that for the matching decomposition algorithm, yielding

$$\mathrm{cost} \leq \beta c\sqrt{b} + o(c\sqrt{b}) + \sqrt{2}\, c \quad (\text{where } b = \left\lceil \tfrac{n}{c^2} \right\rceil).$$

Now $\sqrt{b} < \sqrt{\frac{n}{c^2} + 1} < \sqrt{\frac{n}{c^2}} + 1 = \frac{\sqrt{n}}{c} + 1.$

$$\therefore \mathrm{cost} \leq \beta\sqrt{n} + (\beta+\sqrt{2})\left(\frac{\sqrt{n}}{\sqrt{\mathrm{lglglglg}n}}\right) + o(\sqrt{n})$$

$$= \beta\sqrt{n} + o(\sqrt{n}), \text{ as claimed.}$$

Note that this result is merely of theoretical interest, since one of the "lower order terms" is $\frac{\sqrt{n}}{\sqrt{\mathrm{lglglglg}n}}$ which, for practical purposes is not negligible.

As for the time required, step 2 takes $O(n)$ time, as shown above.

For step 3, an exhaustive search for the shortest tour of r points can be performed in time $r! = O(r^r)$. Therefore the total time required for the calls on the optimizing algorithm is at most

$$\left(\frac{n - \sum_{i=1}^{c^2} b_i}{4b} + c^2\right)(4b)^{4b}$$

$$= O\left(\left(\frac{n}{4b} + c^2\right)(4b)^{4b}\right) = O(c^2(4b)^{4b})$$

(since $\frac{n}{4b} = O(c^2)$).

$$\text{Now } (4b)^{4b} \cdot b^{4b} = (2^b)^8(b^b)^4 \leq (2^b)^8(2^{2^b})^4$$

$$= O[(\mathrm{lglglg}n)^8(\mathrm{lglg}n)^8]$$

$$(\text{since } b \leq \mathrm{lglglglg}n + 1)$$

$$= O(\mathrm{lg}n).$$

$\therefore$ the time needed for step 3 is $O(c^2 \mathrm{lg}n) = O(n\log n)$.

Step 4 can be performed in time $O(c^2 \lg c^2) = O(n\log n)$.

Step 5 takes time $O(n)$, using the method of [3].

Thus, as claimed, the total time is $O(n\log n)$.

SUMMARY

The following tables summarize known results for matching (in both the bounded and unbounded regions) and the traveling salesman problem. Lower order terms are omitted.

Algorithm	Order of running time	Worst case performance ratio
Optimizing	n^3	1
Greedy	$n^2 \log n$	$\frac{4}{3} n^{\lg \frac{3}{2}}$
Spanning Tree	$n^2 \log n$	$\frac{n}{2}$
Hypergreedy without bridges	$n^2 \log n$	$\Omega(n^{\log_3 \frac{3}{2}})$
Hyper-Greedy	$n^2 \log n$	$2 \log_3 n$
Factor of 2 without bridges	$n^2 \log n$	$\Omega(n^{\lg \frac{5}{4}})$
Factor of 2	$n^2 \log K$	8
Factor of 2 with sorting	$n^2(\log n + \log K)$	7

Table 1 Summary of known results for matching n vertices whose distances satisfy the triangle inequality, where K is the ratio of the longest to the shortest edge.

Algorithm	Order of running time	Worst known example cost	Upper bound on worst case cost
Optimizing	n^3	$.537\sqrt{n}$	?
Greedy	$n^2 \log n$	$.806\sqrt{n}$	$1.07\sqrt{n}$
Triangle	$n \log n$	$1.97\sqrt{n}$	$1.97\sqrt{n}$
Rectangle	$n \log n$	$1.44\sqrt{n}$	$1.44\sqrt{n}$
4 Square	$n \log n$	$1.39\sqrt{n}$	?
Square-Rectangle	$n \log n$	$1.5\sqrt{n}$	?
Strip	$n \log n$	$.707\sqrt{n}$	$.707\sqrt{n}$
Decomposition	$n \log n$	same as for optimizing	

Table 2 Summary of known results for matching n vertices in the Euclidean unit square.

Algorithm	Order of running time	Worst known example cost	Upper bound on worst case cost
Optimizing	exponential	$1.07\sqrt{n}$	?
Strip	$n \log n$	$1.41\sqrt{n}$	$1.41\sqrt{n}$
Decomposition	$n \log n$	same as for optimizing	

Table 3 Summary of known results for the traveling salesman problem on n cities in the Euclidean unit square.

REFERENCES

1. Avis, D., "Two Greedy Heuristics for the Weighted Matching Problem, " Proc. Ninth Southeastern Conf. on Combinatorics, Graph Theory, and Computing (1978), 65-76.
2. Avis, D., personal communication.
3. Christofides, N., "Worst-case Analysis of a New Heuristic for the Travelling Salesman Problem," Technical Report of the Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh, PA, 1976.
4. Cornuejols, G. and G. L. Nemhauser, "Tight Bounds for Christofides' Traveling Salesman Heuristic," Math. Prog. 14 (1978), 116-121.
5. Gabow, H., "An Efficient Implementation of Edmonds' Algorithm for Maximum Matching on Graphs," J. ACM 23 (1976), 221-234.
6. Garey, M. R., R. L. Graham, and D. S. Johnson, "Some NP-Complete Geometric Problems," Proc. Eighth Ann. ACM Symp. on Theory of Computing (1976), 10-22.
7. Knuth, D. E., The Art of Computer Programming, Vol. 2: Seminumerical Algorithms, Addison-Wesley Pub. Co., Reading, Mass., 1969.
8. Knuth, D. E., The Art of Computer Programming, Vol. 3: Sorting and Searching, Addison-Wesley Pub. Co., Reading, Mass., 1973.
9. Papadimitriou, C. H., "The Euclidean Traveling Salesman Problem is NP-Complete," Theor. Comput. Sci. 4 (1977), 237-244.
10. Papadimitriou, C. H., "The Probabilistic Analysis of Matching Heuristics," Proc. Fifteenth Annual Allerton Conf. on Communication, Control and Computing (1977), 368-378.
11. Papadimitriou, C. H. and K. Steiglitz, Combinatorial Optimization Algorithms and Complexity, in press.
12. Papadimitriou, C. H., personal communication.
13. Reingold, E. M., J. Nievergelt, and N. Deo, Combinatorial Algorithms: Theory and Practice, Prentice-Hall, Englewood Cliffs, NJ, 1977.
14. Reingold, E. M. and R. E. Tarjan, "On a Greedy Heuristic for Complete Matching," submitted for publication.