Geometry helps in matching
(extended abstract)

Pravin M. Vaidya
AT&T Bell Laboratories,
Murray Hill, NJ 07974.

Abstract

A set of $2n$ points on the plane induces a complete weighted undirected graph as follows: The points are the vertices of the graph and the weight of an edge between any two points is the distance between the points under some metric. We study the problem of finding a minimum weight complete matching (MWCM) in such a graph. We give an $O(n^{2.5}(\log n)^4)$ algorithm for finding an MWCM in such a graph, for the L_1 (*manhattan*), the L_2 (*euclidean*), and the L_∞ metrics. We also study the bipartite version of the problem, where half the points are painted with one color and the other half are painted with another color, and the restriction is that a point of one color may be matched only to a point of another color. We present an $O(n^{2.5}\log n)$ algorithm for the bipartite version, for the L_1, L_2, and L_∞, metrics. The running time for the bipartite version can be further improved to $O(n^2(\log n)^3)$ for the L_1 and L_∞ metrics.

1. Introduction

Given a complete weighted undirected graph on a set of $2n$ vertices, a complete matching is a set of n edges such that each vertex has exactly one edge incident on it. The weight of a set of edges is the sum of the weights of the edges in the set, and a minimum weight complete matching (MWCM) is a complete matching that has the least weight among all the complete matchings.

We study the problem of finding an MWCM in the complete graph induced by a set of $2n$ points on the plane. The points are the vertices of the graph, and the weight of an edge between any two points is the distance between the points under some metric. We shall investigate two common metrics, the L_1 (*manhattan*) metric, and the L_2 (*euclidean*) metric. (We note that the L_∞ metric can be converted to the L_1 metric by rotating the co-ordinate system by forty-five degrees, and so any algorithm for the L_1 metric can be trivially modified to work for the L_∞ metric.) The input consists of $2n$ points which specify the locations of the vertices on the plane. Each point p is given as an ordered pair (p_x, p_y), where p_x and p_y denote the x and y coordinates of p respectively. The distance between two points p and q under the L_r metric is given by $(|p_x-q_x|^r+|p_y-q_y|^r)^{1/r}$. We shall assume that the metric defining the edge weights is fixed.

We also study the bipartite version of the MWCM problem for points on the plane. In the bipartite version, half the points are painted with one color and the other half are painted with another color, and the restriction is that a point of one color can be matched only to a point of the other color.

The complete graph induced by a set of $2n$ points on the plane is entirely specified by the locations of the vertices. So the problem of finding an MWCM in such a graph differs from the problem of finding an MWCM in a general complete graph in that the size of the input is $O(n)$ rather than $\Omega(n^2)$. The input is sparse since the edge weights are implicitly defined by the underlying geometry. It is interesting to investigate if the geometric nature of the MWCM problem for points on the plane can be exploited to obtain an algorithm for its solution which is faster than the $\Theta(n^3)$ algorithm [6, 11] for general graphs. We note that several heuristics for finding a complete matching of small weight (but not necessarily of minimum weight) on points on the plane have been developed [2, 9, 17], but the only known way to find an MWCM on $2n$ points on the plane was to run the MWCM algorithm for general graphs which requires $\Theta(n^3)$ time. In this paper we show that geometry does help to obtain a faster algorithm. We give an $O(n^{2.5}(\log n)^4)$ algorithm for finding an MWCM in the complete graph induced by a set of $2n$ points on the plane, for the L_1 and L_2 metrics. For the bipartite version of the MWCM problem for points on the plane, we give an $O(n^{2.5}\log n)$ algorithm for the L_1 and L_2 metrics. For the bipartite case, the running time of the MWCM algorithm can be further improved to $O(n^2(\log n)^3)$ for the L_1 metric. The space requirement of all the algorithms is $O(n\log n)$.

The algorithms described in the paper will be essentially the well-studied linear programming primal-dual algorithms for weighted matching, namely the Hungarian method [10, 11, 14] for bipartite matching, and Edmond's algorithm [4, 11, 14] for general matching. The primal-dual algorithms for weighted matching associate a dual variable with each vertex of the given graph, and the *slack* associated with an edge is the weight of the edge minus the sum of the dual variables associated with the end vertices of the edge. The algorithms can be substantially speeded up for points on the plane by the application of two key ideas. First, associating a weight with each vertex (point) which is suitably related to the dual variable corresponding to the vertex and which changes much less frequently than the dual variable, and implicitly maintaining the dual variable using the weight. Second, reducing the computation of the minimum slack

for certain subsets of edges to geometric query problems that involve the weights associated with the vertices and that can be efficiently solved using known data structures in computational geometry. Similar ideas can be used to speed up algorithms for related problems like bottle-neck matching [11] for points on the plane, and the transportation problem [11, 14] where the sources and the sinks are sites on the plane and the cost of transporting from a source to a sink is proportional to the distance between the source and the sink.

In section 2 we discuss some geometric query problems that arise naturally in the implementation of the weighted matching algorithm for points on the plane, and sketch solutions to them using using known data structures in computational geometry. In section 3 we give the algorithm for the bipartite version of the MWCM problem for points on the plane. The bipartite case is easier, and serves to illustrate the main ideas that are used in developing the algorithm for the general case. However, the algorithm for the general case is complicated because of certain subsets of vertices of odd cardinality (blossoms [4]) and will be given in the full version of the paper.

We assume a real RAM model of computation [15] standard in computational geometry, so arithmetic operations (i.e. addition, subraction, multiplication, division), memory access operations, and comparison operations, on real numbers require constant time. For the case of the L_2 metric, we make the additional assumption that either square roots can be computed in constant time (so that edge weights can be obtained in constant time) or that the edge weights have been precomputed and are available at the start of the algorithm.

2. Geometric query problems arising in matching on the plane

During the execution of the matching algorithm, we are repeatedly required to compute the minimum slack for certain subsets of edges. To perform this computation efficiently we shall need a good solution to the query problems described below.

Let $d(p, q)$ denote the distance between points p and q. W.r.t. to a set of points P such that there is a weight $w(p)$ associated with each point p in P, we define the following terms. For subsets P_1, P_2 of P, $shortest[P_1, P_2]$ denotes an edge (p_1^*, p_2^*), $p_1^* \in P_1$, $p_2^* \in P_2$, such that

$$d(p_1^*, p_2^*) - w(p_1^*) - w(p_2^*) = \min_{p_1 \in P_1,\, p_2 \in P_2} \{ d(p_1, p_2) - w(p_1) - w(p_2) \}.$$

For a point q, $nearest[q, P]$ denotes a point $p^* \in P$ such that

$$d(q, p^*) - w(p^*) = \min_{p \in P} \{ d(q, p) - w(p) \},$$

and $shortest[q, P]$ denotes the edge $(q, nearest[q, P])$.

Problem 1.

Given a set of points P and a weight $w(p)$ for each point p in P, preprocess P so that for a given query point q, $nearest[q, P]$ can be found quickly.

Problem 2.

We are given a set of points P, an ordering $p_1 < p_2 < \cdots < p_{|P|}$ of the points in P, and a weight $w(p)$ associated with each point p in P. Let $[p_i, p_j)$ be the set of all points p_k in P such that $i \le k < j$. We have to preprocess P, so that given a query point q, and an interval $[p_i, p_j)$ such that $1 \le i < j \le |P|+1$, $nearest[q, [p_i, p_j)]$ can be computed quickly.

In Problems 1 and 2 above the set P is static. We shall also require a solution to the *semi-dynamic* version of Problems 1 and 2. In the semi-dynamic version a new point can be added to P but a point can never be deleted from P. Furthermore, P is totally ordered by the following rule. For a pair of points p, $p' \in P$, $p < p'$ iff p was added to P before p'.

Problem 1 comes up in the bipartite case as well as the general case, and its solution enables us to efficiently compute the minimum slack for various subsets of edges. Problem 2 and the semi-dynamic versions of Problems 1 and 2 arise because of certain subsets of vertices of odd cardinality, called blossoms, in Edmond's algorithm for general weighted matching. One type of blossom corresponds to intervals in some ordering on the set of vertices (points), and given a vertex q and a blossom B of this type, we are required to compute the minimum slack over all edges between q and vertices in B. This leads to Problem 2. The semi-dynamic versions arise because of blossoms merging to form bigger blossoms.

The solutions to the above query problems use segment trees [15], a data structure common in computational geometry. The segment tree for the interval $[i, j)$, i, j integers and $j > i$, is a rooted binary tree defined as follows. The interval $[i, j)$ is associated with the root of the segment tree. If $j > i+1$ then the left subtree is the segment tree for $[i, \lceil \frac{i+j}{2} \rceil)$ and the right subtree is the segment tree for $[\lceil \frac{i+j}{2} \rceil, j)$; if $j = i+1$ then the left and right subtrees are empty. The segment tree data structure extends naturally to an ordered sequence $a_1 < a_2 < \cdots < a_m$ via the correspondence between the interval $[i, j)$ and the interval $[a_i, a_j)$.

For the case of euclidean (L_2) metric the weighted voronoi diagram (WVD) [5, 16] of the points in P provides an adequate solution to Problem 1. Such a voronoi diagram divides the plane into $|P|$ regions (some possibly empty), there being a region $Vor(p)$ for each point $p \in P$. $Vor(p)$ is the region given by

$$Vor(p) = \{ p'' : \forall\, p' \in P,\ d(p'', p) - w(p) \le d(p'', p') - w(p') \}.$$

The WVD of P can be constructed and preprocessed in $O(|P| \log(|P|))$ time, so that given a query point q, in $O(\log(|P|))$ time we can find a point $\bar{p}$ in P such that $q \in Vor(\bar{p})$ [3, 5, 12].

For the case of the L_1 metric we use the Willard-Lueker modification of the two-dimensional range tree [15] to provide a suitable solution to Problem 1. Let $H_x(a, b)$ ($H_y(a, b)$) denote the set of all points p on the plane such that $a \le p_x \le b$ ($a \le p_y \le b$). The range tree (RT) for P is as follows. At the top level is a segment tree for the non-decreasing sequence of the x-coordinates of the points in P. At a segment tree node ψ associated with the interval $[a, b)$ of the x-axis, is stored an ordered list of points in $P \cap H_x(a, b)$, with the points being ordered by y-coordinate. In addition to storing $p \in P \cap H_x(a, b)$ in the ordered list at ψ, we store along with p the points $nearest[(a, p_y), P \cap H_x(a, b) \cap H_y(p_y, \infty)]$, $nearest[(a, p_y), P \cap H_x(a, b) \cap H_y(-\infty, p_y)]$, $nearest[(b, p_y), P \cap H_x(a, b) \cap H_y(p_y, \infty)]$, and $nearest[(b, p_y), P \cap H_x(a, b) \cap H_y(-\infty, p_y)]$. The RT for P can be constructed in $O(|P| \log(|P|))$ time, and using the RT $nearest[q, P]$ can be computed in $O(\log(|P|))$ time for a given point q.

Lemma 1. Given a set of points P on the plane, and a weight $w(p)$ associated with each point p in P, P can be preprocessed in $O(|P|\log(|P|))$ time, so that given a query point q, *nearest*$[q, P]$ and *shortest*$[q, P]$ can be found in $O(\log(|P|))$ time. ■

The data structure for Problem 2 has two levels. At the top level is a segment tree for the ordered sequence $p_1 < p_2 < \cdots < p_{|P|}$ of the points in P, and at a segment tree node associated with the interval $[p_k, p_l)$ is stored the WVD for the set $[p_k, p_l)$ in the case of the L_2 metric, and the RT for the set $[p_k, p_l)$ in the case of the L_1 metric.

Lemma 2. Let $P = \{p_1 < p_2 < \cdots < p_{|P|}\}$ be an ordered set of points on the plane, and let there be a weight $w(p)$ associated with each point p in P. P can be preprocessed in $O(|P|(\log(|P|))^2)$ time, so that given a query point q and an interval $[p_i, p_j)$ such that $1 \le i < j \le |P|+1$, *nearest*$[q, [p_i, p_j)]$ and *shortest*$[q, [p_i, p_j)]$ can be found in $O((\log(|P|))^2)$ time. ■

Using standard techniques [13] the above mentioned static data structures for Problems 1 and 2 can be converted into semi-dynamic data structures to allow for insertions into P. The semi-dynamization increases the query time and the amortized time per insertion by a factor of at most $2\lceil \log_2 P \rceil$.

3. Weighted bipartite matching on the plane

We are given two sets U and V each consisting of n points on the plane. U and V induce a complete bipartite graph whose vertices are the points in U and V, and the weight of an edge (u_i, v_j), $u_i \in U$, $v_j \in V$, is the distance between u_i and v_j under some metric. We consider the L_1 and the L_2 metrics. The problem is to find a minimum weight complete matching in the complete bipartite graph on U and V.

Let $u_1, \cdots, u_n$ be an enumeration of the vertices in U, and let $v_1, \cdots, v_n$ be an enumeration of the vertices in V. The Hungarian method [10, 11, 14] for weighted bipartite matching associates dual variables α_i and β_j with vertices u_i and v_j respectively. A *feasible* matching consists of a matching M and dual variables α_i and β_j such that

$$\alpha_i + \beta_j \le d(u_i, v_j), \quad 1 \le i \le n, \ 1 \le j \le n$$

$$\alpha_i + \beta_j = d(u_i, v_j), \quad (u_i, v_j) \in M$$

A feasible matching which is complete is a minimum weight complete matching [11, 14]. We start with dual variables $\beta_j = \min_i \{d(u_i, v_j)\}$, $1 \le j \le n$, and $\alpha_i = 0$, $1 \le i \le n$. So the empty matching is initially feasible. The method proceeds in phases and during each phase the feasible matching M is augmented by an edge. So there are n phases.

A vertex is *exposed* if it is not matched by an edge in the current matching M. An *alternating path* is a path that alternately traverses an edge in M, and an edge not in M. An *augmenting path* is an alternating path between two exposed vertices. An edge (u_i, v_j) is *admissible* iff $\alpha_i + \beta_j = d(u_i, v_j)$. A phase consists of searching for an augmenting path among admissible edges.

For each exposed vertex in V, we grow an alternating tree rooted at the vertex. Each vertex in $U \cup V$ which is in an alternating tree is reachable from the root of the tree via an alternating path that uses only admissible edges. S (T) denotes the set of all vertices $v \in V$ ($u \in U$) such that v (u) is in an alternating tree. Let $F = U - T$. At the beginning of a phase S consists of the exposed vertices in V, and $F = U$. Let δ be defined as

$$\delta = \min_{u_i \in F,\ v_j \in S} \{d(u_i, v_j) - \alpha_i - \beta_j\}.$$

We use a variable Δ to keep track of the sum of dual changes δ, and associate a weight $w(v)$ ($w(u)$) with each vertex v in V (u in U). The weights are used to implement a phase efficiently. At the beginning of a phase $\Delta = 0$, for each $u_i \in U$, $w(u_i) = \alpha_i$, and for each $v_j \in V$, $w(v_j) = \beta_j$. Depending on whether δ equals zero or exceeds zero, the alternating trees are grown or there is a dual variable change.

Case 1. $\delta = 0$, and (u_i, v_j), $u_i \in F$, $v_j \in S$, is admissible.

If u_i is exposed then construct an augmenting path among the admissible edges by backtracking from v_j to the root and augment M, and the phase ends;

If u_i is matched to v_k then

$F := F - \{u_i\}$, $T := T \cup \{u_i\}$, $S := S \cup \{v_k\}$,

$w(u_i) := \alpha_i + \Delta$, $w(v_k) := \beta_k - \Delta$. ■

Case 2. $\delta > 0$.

$\Delta := \Delta + \delta$;

For each vertex $u_i \in T$, $\alpha_i := \alpha_i - \delta$;

For each vertex $v_j \in S$, $\beta_j := \beta_j + \delta$. ■

Note that for each vertex $v_j \in S$, β_j equals $w(v_j) + \Delta$, for each vertex $u_i \in T$, α_i equals $w(u_i) - \Delta$, and for each vertex in F its dual variable equals its weight. So it suffices to update Δ, and the weights associated with the vertices, rather than explicitly updating the dual variables α_i, β_j. At the end of a phase the correct values of the dual variables may be computed using Δ and the weights. Using the relation between the weights and the dual variables we may write

$$\delta = \min_{u \in F,\ v \in S} \{d(u, v) - w(u) - w(v)\} - \Delta.$$

We shall use the data structures used in the solution of Problem 1 in section 2, namely the weighted voronoi diagram (WVD) and the range tree (RT), to efficiently compute δ, and an edge (u, v), $u \in F$, $v \in S$, such that $d(u, v) - w(u) - w(v) = \delta + \Delta$. Throughout a phase, S is partitioned into S_1 and S_2 such that $|S_2| \le \sqrt{n}$. Also, F is partitioned into $F_1, F_2, \cdots, F_{\lceil n^{0.5} \rceil}$, (some of the F_i's possibly empty) such that $|F_i| \le \lceil n^{0.5} \rceil$, $1 \le i \le \lceil n^{0.5} \rceil$. We maintain the following data structures.

1. A priority queue containing the edge *shortest*$[u, S_1]$ for each $u \in F$. The priority of an edge (u, v) in this queue is $d(u, v) - w(u) - w(v)$.
2. A priority queue containing the edges *shortest*$[v, F_i]$, $1 \le i \le \lceil n^{0.5} \rceil$, for each vertex v in S_2. The priority of an edge (u, v) in this queue is also $d(u, v) - w(u) - w(v)$.
3. The WVD/RT for each of the sets $F_1, F_2, \cdots, F_{\lceil n^{0.5} \rceil}$.

δ and an edge for which δ is achieved can be obtained in $O(\log n)$ time by examining an edge with minimum priority in 1 and 2 above. Vertices added to S are always inserted into S_2. In order to maintain the condition that $|S_2| \le \sqrt{n}$, whenever the size of S_2 reaches the threshold of $\sqrt{n}$ we add all the vertices in S_2 to S_1 and reset S_2 to the null set. Then *nearest*$[u, S_1]$ must be recomputed for every $u \in F$. From Lemma 1 in section 2, this recomputation may be done in $O(n \log n)$ time using a WVD/RT for S_1, leading to total of $O(n^{1.5} \log n)$ operations for the recomputations in a phase.

An insertion into S_2 and a deletion from F each require $O(n^{0.5} \log n)$ operations. Suppose a vertex v is inserted into S_2. Then for each F_i, $1 \le i \le \lceil n^{0.5} \rceil$, *shortest*$[v, F_i]$ is computed in $O(\log n)$ time using the WVD/RT for F_i. Hence an insertion costs $O(n^{0.5} \log n)$ operations. Suppose a vertex u is deleted from F, and suppose $u \in F_i$. Then the WVD/RT for F_i is recomputed, and *shortest*$[v, F_i]$ is also recomputed for all the vertices v in S_2. So a deletion costs $O(n^{0.5} \log n)$ operations.

Since there are $O(n)$ insertions into S_2 and $O(n)$ deletions from F, a phase takes $O(n^{1.5}\log n)$ time. This gives a total running time of $O(n^{2.5}\log n)$ for the weighted bipartite matching algorithm for points in the plane. In the full paper we show that the running time of the bipartite matching algorithm can be further improved to $O(n^2(\log n)^3)$ for the L_1 metric.

4. Weighted general matching on points in the plane

In the full paper we shall describe an $O(n^{2.5}(\log n)^4)$ algorithm for finding an MWCM in the complete graph induced by $2n$ points on the plane. Two things are crucial to the running time of the algorithm: (i) associating weights with vertices and blossoms which are suitably related to the dual variables and which change much less frequently than the corresponding dual variables, and implicitly maintaining the dual variables using the weights, (ii) reducing the computation of the minimum slack for certain subsets of edges to geometric query problems (described in section 2) that involve the weights associated with the vertices.

5. Conclusion

We have shown that the underlying geometry can be exploited to speed up algorithms for weighted matching when the vertices of the graph are points on the plane and the weight of an edge between two points is the distance between the points under some metric. The techniques described in the paper can be used to speed up algorithms for related problems like bottleneck matching [11] for points on the plane, and the transportation problem [11, 14] where the sources and the sinks are located on the plane and the cost of transporting from a source to a sink is proportional to the distance between the source and the sink. The techniques in the paper can also be utilized to speed up scaling algorithms [7, 18] for matching and related problems by a factor of about $\sqrt{n}$ for points on the plane. Finally, we note that for the L_1 and the L_∞ metrics the algorithms in the paper easily extend to the case where the vertices of the graph are points in d-dimensional space (d fixed) rather than points on the plane. For points in d-dimensional space we use d-dimensional range trees instead of 2-dimensional range trees [15], and this increases the running time of the matching algorithms by at most $O(\log n)^d$.

Acknowledgements

The author would like to thank F. P. Preparata for suggesting the problem of matching on the plane. The author like to thank H. Edelsbrunner, S. Fortune, and P. Shor, for helpful discussions. The author would like to thank M. R. Garey and D. S. Johnson for helpful suggestions concerning the presentation of this paper.

References

1. A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, Mass. 1974.
2. D. Avis, A survey of heuristics for the weighted matching problem, *Networks*, 13, 1983, pp. 475-493.
3. H. Edelsbrunner, L. J. Guibas, and J. Stolfi, Optimal point location in a monotone subdivision, Technical Report, DEC Systems Research Center, 1984.
4. J. Edmonds, Maximum matching and a polyhedron with 0,1-vertices, *J. Res. Nat. Bur. Standards*, 69B, 1965, 125-130.
5. S. Fortune, A sweepline algorithm for voronoi diagrams, *Proc. ACM Annual Symp. Computational Geom.*, 1986, 313-322.
6. H. N. Gabow, An efficient implementation of Edmond's algorithm for maximum matching on graphs, *J. ACM*, 23, 1976, 221-234.
7. H. N. Gabow, and R. E. Tarjan, Faster scaling algorithms for network problems, Technical Report, 1987, Dept. of Comp. Sc., Princeton University.
8. Z. Galil, S. Micali, and H. N. Gabow, Priority queues with variable priority and an $O(EV\log V)$ algorithm for finding a maximal weighted matching in general graphs, *Proc. 22nd Annual IEEE Symp. Found. Comp. Sc.*, 1982, pp.255-261.
9. M. Iri., M. Murota, and S. Matsui, Linear time heuristics for minimum-weight perfect matching on a plane with application to the plotter problem, Unpublished, 1980.
10. H. W. Kuhn, The Hungarian method for the assigment problem, *Naval Res. Logistics Quart.*, 2, 1955, pp. 83-97.
11. E. Lawler, *Combinatorial Optimization: Networks and Matroids*, 1976, Holt Rinehart and Winston, New York, 1976.
12. D. T. Lee, and F. P. Preparata, Location of a point in a planar subdivision and its applications, *SIAM J. Comput.*, Vol. 6, 1977, pp. 594-606.
13. K. Mehlhorn, *Data Structures and Algorithms 3: Multidimensional Searching and Computational Geometry*, pp. 2-9, EATCS Monographs on Theoretical Computer Science, Springer-Verlag, 1984.
14. C. H. Papadimitriou, and K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity*, 1982, Prentice-Hall, Inc., Englewood Cliffs, NJ.
15. F. P Preparata, and M. I. Shamos, *Computational Geometry: An Introduction*, 1985, Springer-Verlag New York Inc.
16. M. Sharir, Intersection and closest-pair problems for a set of planar discs, *SIAM J. Comput.*, Vol. 14 (2), May 1985, pp. 448-468.
17. K. J. Supowit, and E. M. Reingold, Divide-and-Conquer heuristics for minimum weighted euclidean matching, *SIAM J. Comput.*, Vol 12. No. 1, 1983, pp. 118-144.
18. H. N. Gabow, and R. E. Tarjan, Faster scaling algorithms for graph matching, Tech. Report, Dept. of Computer Science, Princeton University, 1987.