Efficient Minimum Cost Matching Using Quadrangle Inequality

Alok Aggarwal * Amotz Bar-Noy* Samir Khuller † Dina Kravets ‡
Baruch Schieber*

Abstract

We present efficient algorithms for finding a minimum cost perfect matching, and for solving the transportation problem in bipartite graphs, $G = (Red \cup Blue, Red \times Blue)$, where $|Red| = n$, $|Blue| = m$, $n \leq m$, and the cost function obeys the quadrangle inequality. Our first results assume that all the red points and all the blue points lie on a curve that is homeomorphic to either a line or a circle and the cost function is given by the Euclidean distance along the curve. We present a linear time algorithm for the matching problem that is simpler than the one in [KL75]. We generalize our method to solve the corresponding transportation problem in $O((m+n)\log(m+n))$ time, improving on the previously known algorithm in [KL75].

Our next result is an $O(n \log m)$ algorithm for minimum cost matching when the cost array is a bitonic Monge array. An example of this is when the red points lie on one straight line and the blue points lie on another straight line (that is not necessarily parallel to the first one). Finally, we provide a weakly polynomial algorithm for the transportation problem in which the associated cost array is a bitonic Monge array. Our algorithm for this problem runs in $O(m \log(\sum_{i=1}^{m} a_i))$ time, where a_i is the supply available at the ith source, b_j is the demand at the jth sink, and $\sum_{i=1}^{m} a_i \geq \sum_{j=1}^{n} b_j$.

*IBM Research Division, T. J. Watson Research Center, Yorktown Heights, NY 10598.
Email: {aggarwa,amotz,sbar}@watson.ibm.com.
†Dept. of Computer Science, University of Maryland, College Park, MD 20742.
Email: samir@cs.umd.edu. Supported in part by NSF grants CCR-8906949, CCR-9111348 and NSF-9103135. Part of this research was done while this author was visiting the IBM T. J. Watson Research Center.
‡Dept. of Computer Science, New Jersey Institute of Technology, University Heights, Newark, NJ 07102.
Email: dina@cis.njit.edu. Supported while at MIT, in part by the Air Force under Contract AFOSR-89-0271, the Defense Advanced Research Projects Agency under Contracts N00014-87-K-825 and N00014-89-J-1988.

1 Introduction

Given a complete bipartite graph $G = (Red \cup Blue, Red \times Blue)$, where $|Red| = n$, $|Blue| = m$ and $n \leq m$, a *perfect matching* on G is a subset of the edges $M \subseteq E$ such that for all red nodes p, exactly one edge of M is incident to p and for all blue nodes q, at most one edge of M is incident to q. Given a cost function defined on the edges of G, a *minimum cost perfect matching* on G is a perfect matching on G of minimum cost.

The best known algorithm for this problem for arbitrary bipartite graphs (with $m+n$ nodes and mn edges) takes $O(n(mn + m\log m))$ time. This time complexity can be achieved by using the Hungarian method [Kuh55, AMO89, GTT89]. Since this problem has a relatively high time complexity, researchers have investigated special cases. For example, Vaidya [Vai89] shows that the minimum cost perfect matching among $2n$ points in the Euclidean plane can be computed in $O(n^{5/2}\log^4 n)$ time. For the case where $n = m$, the red and blue points lie on a convex polygon (respectively, simple polygon), and the distance between two points is simply the Euclidean distance, Marcotte and Suri [MS91] show that a minimum cost matching can be computed in $O(n \log n)$ time (respectively, $O(n \log^2 n)$ time). Note that these papers assume that $n = m$, in spite of the fact that the situation $n < m$ arises in many matching problems. The only paper that considers the $n \leq m$ case is by Karp and Li [KL75]. They investigated the case where the points lie on either a line or a circle, and the cost function is given by the Euclidean distance along the corresponding curve. For this problem, they obtained a linear time algorithm for minimum weight matching, assuming that the points are given in sorted order. (See also [WPMK86].) In this paper, we continue the investigation along the lines of [KL75, WPMK86, Vai89, MS91]. As in [KL75], we do *not* assume that n equals m.

For the transportation problem we assume that each blue point i has supply a_i, and each red point j has demand b_j, where $\sum_{i=1}^{m} a_i \geq \sum_{j=1}^{n} b_j$. A *fea-*

0-8186-2900-2/92 $3.00 © 1992 IEEE

sible transportation is an assignment of supplies from blue points to red points such that all the demands are satisfied. The cost of moving one unit of supply from a blue point to a red point is given by the cost function defined on the edges. A ***minimum cost transportation*** is a feasible one with minimum cost.

Hoffman [Hof63], following the French mathematician Gaspard Monge (1746–1818), considered a special case of the transportation problem in which the cost array forms an $n \times m$ *Monge* array. (The entry $a[i,j]$ of the cost array is the cost of the edge between red point i and blue point j.) An $n \times m$ array $A = \{a[i,j]\}$ is *Monge* if for all i_1, i_2, j_1, and j_2 satisfying $1 \leq i_1 < i_2 \leq n$ and $1 \leq j_1 < j_2 \leq m$,

$$a[i_1,j_1] + a[i_2,j_2] \leq a[i_1,j_2] + a[i_2,j_1] .$$

Hoffman [Hof63] shows that if the cost array associated with a transportation problem is Monge and the total supply is the *same* as the total demand, then a simple greedy algorithm solves the transportation problem in linear time. However, in reality, a more reasonable assumption is that the total supply is greater than or equal to the total demand. In this paper we consider special cases of the transportation problem in which the total supply may be greater than the total demand.

The remainder of this section lists the main results of this paper and their applications.

Our first results are for case where the red and blue points lie on a curve that is homeomorphic to either a line or a circle and the cost function is the Euclidean distance along the curve. For this case, we give a linear time algorithm for computing the minimum cost matching that is simpler than the one given by Karp and Li [KL75]. Our algorithm can be extended to the corresponding transportation problem, where each blue point has integral supply and each red point has integral demand. (We assume that the sum of the supplies exceeds the sum of the demands.) For this transportation problem we develop an algorithm with an $O((m+n)\log(m+n))$ running time. To achieve this bound, we exploit some underlying properties of the matching. This improves on the $O((m+n)^2)$ time algorithm of Karp and Li [KL75].

Our matching algorithm for the points on the circle can also be applied to the case where the red and the blue points form the vertices of a convex *unimodal* polygon [AM86]. A polygon is *unimodal* if for every vertex, the distances from it to all other vertices form a sequence that is first non-decreasing and then non-increasing when the polygon is traversed in clockwise order starting at that vertex. As an application, we note that the case when the points lie on the circle arises in pattern recognition when feature sets of different objects are compared. We refer the reader to [WPR84, WPMK86] for a detailed discussion of this application.

Our next result is for the case when the cost array forms an $n \times m$ bitonic *Monge* array. An array is bitonic if the entries in every row form a monotone non-increasing sequence that is followed by a monotone non-decreasing sequence. For the bitonic Monge array, we present an $O(n \log m)$ time algorithm for computing a minimum cost matching. Notice that the cost array for the sorted red and blue points on a line forms a bitonic Monge array[1]. As a matter of fact, this is true even for a more general case where the red points lie on one straight line and the blue points lie on another straight line (that is not necessarily parallel to the first one). Hence this result can be applied to the problem of connecting power lines (or connecting clock lines) in VLSI river routing [MC80, Won91]. In this problem, n terminals that lie on a straight line need to be connected to any n of the m power (or clock) terminals that lie on a parallel line, and the total amount of wire used should be minimized. (Another approach is to construct a Steiner tree, but then it is harder to accommodate current density limitations or the clock-skew, and therefore this approach is rarely used.)

Our results for bitonic Monge arrays can be applied to the transportation problem. We provide a weakly polynomial algorithm for the transportation problem when the associated cost array is a bitonic Monge array. Our algorithm for this problem runs in $O(m \log(\sum_{i=1}^{m} a_i))$ time where a_i is the supply available at the ith source, b_j is the demand at the jth sink, and $\sum_{i=1}^{m} a_i \geq \sum_{j=1}^{n} b_j$.

Finally, we remark that the following problem appeared in an exam for an algorithms course taught by C. E. Leiserson [Lei91]: given m pairs of skis with heights $s_1, \ldots, s_m$ and $n \leq m$ skiers with heights $t_1, \ldots, t_n$, assign skis to skiers so that the sum of the absolute differences of the heights of each skier and his/her skis is minimized. Observe that there is a fairly simply $O(nm)$ dynamic programming algorithm for this problem (which we believe was the intent of the problem in the exam). The algorithm by Karp and Li [KL75] as well as our algorithm solve this problem in $O(m \log m)$ time.

All our results use the quadrangle inequality in a crucial manner. Because of the many applications,

[1]Observe that the cost array for the points on a circle is *not* bitonic Monge.

we hope that this paper will generate more interest towards the understanding of minimum cost bipartite matching and transportation for useful special cases. For example, the three important problems that we were not able to solve are: (i) An efficient computation of the minimum cost matching when the cost array is (staircase) Monge; a special case of this is when the red and blue points form a convex polygon. (ii) An efficient computation of the minimum cost matching when the red and blue points form a simple polygon. (iii) The transportation problem when the underlying cost array is Monge and the total supply is greater than the total demand.

2 Minimum matching on a circle

In this section we present our algorithm for the minimum cost perfect matching problem for the case where the red points and the blue points lie on a circle and the cost function is the Euclidean distance along the arcs of the circle. For the sake of simplicity, we assume that all the points are distinct. The algorithm and all the proofs are analogous for points on the line. Furthermore, these results hold for points on any curve homeomorphic to a circle or a line, as long as the cost function is the distance along the curve. The results also hold for the case where all the points are vertices of a convex unimodal polygon and the cost is the Euclidean distance between points.

Consider a circle with red and blue points. Any two points p and q split the circle into two arcs (see Figure 1), one going clockwise from p to q, and one going counterclockwise from p to q. Let $cw(p,q)$ (*respectively*, $ccw(p,q)$) be the clockwise (respectively, counterclockwise) arc from p to q. Let $x(p,q)$ refer to the shorter of $cw(p,q)$ and $ccw(p,q)$. The distance between p and q, $d(p,q)$, is defined to be the length of $x(p,q)$.

We say that two edges in the matching M, $(p,q) \in M$ and $(p',q') \in M$, *cross* each other if $x(p,q) \cap x(p',q') \neq \emptyset$, $x(p,q) \not\subseteq x(p',q')$, and $x(p',q') \not\subseteq x(p,q)$. We call a matching *nested* if no two edges in the matching cross each other. In other words, a matching is nested if for any two edges in the matching: $(p,q) \in M$ and $(p',q') \in M$, $x(p,q) \cap x(p',q') \neq \emptyset$ implies that either $x(p,q) \subseteq x(p',q')$ or $x(p',q') \subseteq x(p,q)$.

Lemma 1 *There exists a nested minimum cost perfect matching for points on a circle.*

Proof: Let M be any minimum cost perfect matching. We show how to uncross any pair of crossed edges of M to get another minimum cost perfect matching with fewer edge crossings. Let $(p,q) \in M$ and $(p',q') \in M$ be some crossed pair of edges, where $p,p' \in Red$ and $q,q' \in Blue$. Consider a matching $M' = M - \{(p,q),(p',q')\} + \{(p,q'),(p',q)\}$, which is simply the matching M with (p,q) and (p',q') uncrossed. First, we show that the cost of M' is less than or equal to the cost of M.

Case 1: $q' \in x(p,q)$ and $q \in x(p',q')$ (see Figure 2 (a)). We obtain a contradiction by showing that M' costs less than M. This is because $d(p,q')+d(p',q) = [d(p,q)-d(q',q)]+[d(p',q')-d(q',q)] < d(p,q)+d(p',q')$.

Case 2: $p' \in x(p,q)$ and $q \in x(p',q')$ (see Figure 2 (b)). Here we show that the cost of M' is at most the cost of M.

$$\begin{aligned} &d(p,q)+d(p',q') \\ &\quad = [d(p,p')+d(p',q)]+[d(p',q)+d(q,q')] \\ &\quad = [d(p,p')+d(p',q)+d(q,q')]+[d(p',q)] \\ &\quad \geq d(p,q')+d(p',q). \end{aligned}$$

We now argue that whenever M' costs the same as M in Case 2, then M' has fewer edge crossings than M. In particular, we show that if some edge $(s,t) \in M$ crosses either (p',q) or (p,q'), it must have crossed either (p,q) or (p',q'). This completes the proof, since we have uncrossed (p,q) and (p',q') and, hence, the number of edge crossings in M' must be at least one fewer that in M. The following case analysis proves the above claim.

Case 2.1: (s,t) crosses only (p',q), i.e. $s,t \in cw(p,q')$. W.l.o.g, assume $s \in cw(p',q)$ and $t \in ccw(p',q)$. If $t \in cw(p,p')$, then (s,t) crosses (p',q'). Otherwise, $t \in cw(q,q')$ and (s,t) crosses (p,q).

Case 2.2: (s,t) crosses only (p,q'), i.e. $s,t \notin cw(p',q)$. W.l.o.g, assume $s \in cw(p,q')$, and $t \in ccw(p,q')$. If $s \in cw(p,p')$, then (s,t) crosses (p,q). Otherwise, $s \in cw(q,q')$ and (s,t) crosses (p',q').

Case 2.3: (s,t) crosses both (p',q) and (p,q'). W.l.o.g, assume $s \in cw(p',q) \cap cw(p,q') = cw(p',q)$ and $t \in ccw(p,q') \cap ccw(p',q) = ccw(p,q')$. In this case (s,t) crosses both (p,q) and (p',q').

■

For $p \in Red$, define the ***left partner*** (respectively, ***right partner***) of p, denoted by ℓ_p (respectively, r_p), to be the first blue point q counterclockwise (respectively, clockwise) from p, such that going counterclockwise (respectively, clockwise) from p to q on the circle, there are *as many* red points *as* blue points.

Lemma 2 *In a nested minimum cost perfect matching, every red point is matched to either its left partner or its right partner.*

Proof: (By contradiction.) W.l.o.g, suppose that $(p,q) \in M$, $x(p,q) = cw(p,q)$, and $q \neq r_p$.

Case 1: $r_p \notin x(p,q)$ (see Figure 3 (a)). From the definition of the right partner, there must be a red point $p' \in x(p,q)$ such that $(p',q') \in M$ and $q' \notin x(p,q)$. Then (p,q) crosses (p',q'), a contradiction.

Case 2: $r_p \in x(p,q)$ (see Figure 3 (b)). Since there is an equal number of red and blue points in $x(p,r_p)$, one blue point q' in $x(p,r_p)$ or r_p itself, is either not matched or matched with some p' not in $x(p,r_p)$. If q' is not in M, then $M - \{(p,q)\} + \{(p,q')\}$ costs less than M, a contradiction. If $(p',q') \in M$, then $p' \in x(r_p,q)$ since M is nested. Therefore, $M - \{(p,q),(p',q')\} + \{(p,q'),(p',q)\}$ gives a cheaper matching than M, a contradiction.

■

Define the chain $C = \langle v_0, u_1, v_1, u_2, v_2, \ldots, u_k, v_k \rangle$ to be an *alternating chain* of points if the following conditions hold:

1. $\{u_1, u_2, \ldots, u_k\} \subseteq \mathit{Red}$ and $\{v_0, v_1, v_2, \ldots, v_k\} \subseteq \mathit{Blue}$;
2. the blue point v_0 is the left partner of u_1 and the blue point v_k is the right partner of u_k;
3. the blue point v_i is the left partner of u_{i+1} and the right partner of u_i, for each $1 \leq i \leq k-1$; and
4. the chain C is maximal.

Note that since C is maximal, either v_0 is not a right partner and v_k is not a left partner of any red point, or v_0 is the same as v_k.

Observe that in the case of points on a line, a chain may not necessarily start at a blue point.

Lemma 3 *Each red point belongs to a unique alternating chain, and any two chains are disjoint.*

Proof: To see why the lemma is true, observe the following property. Consider a blue point v_i. We claim that only *one* red point can have v_i as its left partner. Suppose that two red points r_a and r_b have v_i as their left partner. Assume that we have v_i, r_a, r_b in clockwise order. Between r_a and v_i there is an equal number of red and blue points. If v_i is the left partner of r_b, then due to the presence of r_a (a red point) there must be an excess blue point between r_a and r_b. In this case, this blue point must be the left partner of r_b, yielding a contradiction. The same property is true for right partners. With this observation it is easy to see why the lemma is true. ■

The following theorem is implied by Lemma 2 and Lemma 3 and is the key for our algorithm.

Theorem 4 *Consider the points of a single chain in a nested minimum cost matching. There is one unmatched blue point in the chain. All the red points to the left of this blue point are matched to their left partners, and the red points to the right of this blue point are matched to their right partners.*

Note that the theorem holds when the chain is a cycle, i.e., $v_0 = v_k$. In this case, the unmatched blue point in the chain is either v_0 or v_k.

The algorithm

Step 1: Sort all the blue and red points (together) in some direction around the circle, say clockwise.

Step 2: Compute ℓ_p and r_p for each red point p. We show how to compute the r_p's using a stack (finding the ℓ_p's is analogous). Starting with any red point, we proceed clockwise around the circle. Any time we encounter a red point, we push it onto the stack. Any time we encounter a blue point q, we pop the last red point p from the stack and set $q = r_p$. If at any point the stack becomes empty, we push the clockwise-next red point onto it, ignoring any blue points that this action causes us to skip.

Step 3: Compute all the alternating chains. This can be done by keeping links between the red points and their partners. Starting with any red point that is not already contained in a chain, we simply trace the left and right links until we encounter the blue endpoints of the chain.

Step 4: For each alternating chain $C = \langle v_0, u_1, v_1, u_2, v_2, \ldots, u_k, v_k \rangle$, decide which blue point is free as follows. We compute M_i^C, which is the cost of the matching with the blue point v_i being unmatched. We first compute M_1^C (this is easy), and then compute M_{i+1}^C from M_i^C. This involves changing the matched edge (u_{i+1}, v_{i+1}) to (u_{i+1}, v_i). It takes $O(k)$ time to compute M_1^C (where k is the length of the chain), and $O(k)$ time to compute all the other M_i^Cs. If i^* is the index that gives the smallest value for M_i^C, the points u_i on C with $i \leq i^*$ match to their left partners, and the others match to their right partners.

The correctness of this algorithm essentially follows from Theorem 4. As for time complexity, it is not hard to verify that each step, except for sorting the points, can be done in linear time. We obtain the following theorem.

Theorem 5 (Complexity) *The algorithm runs in linear time if the points are given sorted in some direction along the circle.*

Our algorithm can be parallelized. The parallel time complexity is $O(\log m)$ time with m processors on an EREW PRAM.

3 Transportation problem on a line

We now consider the transportation problem, where the sources and sinks are on a straight line, with integer supplies and demands. The algorithm for this problem runs in $O((m+n)\log(m+n))$ steps where the number of sources (*blue* nodes) is m, and the number of sinks (*red* nodes) is n. We assume that a_i is the integer supply at the ith source node (blue node), and b_j is the integer demand at the jth sink node (red node), and that $\sum_{i=1}^{m} a_i \geq \sum_{j=1}^{n} b_j$. Due to lack of space we only outline the high-level ideas behind this algorithm. We describe the algorithm for the line, since it is simpler, and we note that the algorithm for the circle is very similar.

As in the case for the red/blue points on the line, we proceed to define *left partners* and *right partners* for the red points. We view each red node j as a cluster of b_j red points, and each blue node i as a cluster of a_i blue points. (Keep in mind the distinction between *nodes* and *points.*) Left partners and right partners of the red points are now defined exactly as before. We can assume that in each cluster the points are placed infinitesimally close to each other. Since there is a large number of red/blue points, we do not compute the partners explicitly; but these are maintained implicitly for clusters of points, and can be implicitly computed by keeping track of the supplies and demands of the nodes as they are pushed and popped from the stack. This takes time linear in the number of nodes. Each blue point p has at most one left partner and one right partner (as before). In other words, at most one red point from the left and one red point from the right may choose p as its partner.

Each blue node is now "split" into sub-nodes. Blue points p and q are in the same sub-node, if ℓ_p and ℓ_q are red points corresponding to the same red node, *and* r_p and r_q are red points corresponding to the same red node. This gives us the property that each blue sub-node can have at most one left partner and at most one right partner. The number of blue sub-nodes could exceed m (the number of blue nodes), but is $O(m+n)$. From now on we will refer to *both* nodes and sub-nodes simply as nodes. As before, we have the property that in any minimum cost solution to the transportation problem each red point is supplied by either its left partner or its right partner.

We briefly review the main steps of the algorithm. The input to this algorithm is the set of nodes, along with the location and demand/supply of each node. We also assume that the left and right partners of each blue node have been computed.

A chain C, as before, is an *alternating chain* of points. The leftmost point of a chain is called the root of the chain. We define a "bundle of chains" to be a maximal collection of chains that visit the same nodes (and also start and end at points in the same nodes). The *width* of each bundle of chains is the number of chains in that bundle. The leftmost node of the bundle of chains is called the root of the bundle.

Bundles of chains that start with a red node and end with a blue node can be handled by a scan of the nodes. Consider a red node i, with demand b_i. Suppose ℓ_i of the points in node i have left partners, and r_i of the points have right partners. Clearly, $r_i + \ell_i \geq b_i$ (since there are more supply points than demand points). The rightmost $(b_i - \ell_i)$ points in i's cluster have no left partners and must match to their right. These are chains starting at a red node, and we match these red points to the right. This will match off some of the blue points, thus leaving other red points without left partners — starting new chains. In a scan of the red points from left to right we can handle all chains that have a red left endpoint. Similarly, in a scan of the red points from right to left we can handle all the chains with a red right endpoint.

The difficult case is in handling bundles of chains that start and end with blue nodes (and processing them all simultaneously). In this situation the bundle of chains starts and ends with a blue node, and the chains could match in any direction.

We now process the blue nodes from left to right. During the execution of the algorithm, corresponding to each blue node we have a set of bundles going through the blue node. The set will be stored in a data-structure that will enable the required operations to be done efficiently. Each element in the set is a bundle that goes through that blue node with the following information:

1. The "shifting" cost of the bundle of chains — the total length of the edges from red nodes to their left partners minus the total length of the edges

from red nodes to their right partners.

2. The root node of the bundle of chains.

3. The width of the bundle (number of chains).

Bundles of chains with negative shifting costs clearly want to match to their left. Clearly, if for a chain with the least shifting cost in the set we decide to match it to the right, then we will decide the same for the rest of the chains in the set.

The algorithm scans blue nodes from left to right. In this scan, if at any time a red point decides to match to its left, it can be matched off, since it does not affect any future matches. We now describe the computation that is done when we scan over a blue node p.

Step 1: If the set of bundles of p is not empty, then remove the elements with the smallest shifting costs from the set as long as these shifting costs are negative. These bundles of chains prefer to match to their left, and can be matched off. This releases the corresponding blue points in p's cluster that are on the chains just matched to the left. These blue points are going to be roots of new chains since they have no left partners.

Step 2: If the set of bundles of p is not empty, the remaining elements in the set all have a positive shifting cost, and *so far* prefer to match to their right. The blue successor of a blue *point* q is defined to be the right (blue) partner of the right (red) partner of q. If blue successors exist, we continue to extend these chains by examining the blue successors of the blue points in the current set. We put an element in the set of each node that contains blue successors of the blue points that are in the current set. This involves locating the sets to insert elements into, and inserting them efficiently into the sets by spending time proportional to the number of nodes that contain blue successors (and not in the number of chains in the current set). Notice that for this step we also need to update the shifting costs of the chains that are inserted in the set of any particular blue node; but all these are changed by the same amount. Moreover, all the elements that are being inserted have smaller shifting costs than the elements already in the set. For the points that have no blue successors, we have reached the end of the chains, and we match the red points in these chains to the right.

Step 3: Create new elements corresponding to the points in p that have no left partners. (Some of these may be the points that were made free in Step 1.) We examine the blue successors of these blue points, and put an element in the set of each node that contains blue successors of these blue points. The shifting cost of these bundles of chains is easy to compute. Clearly, these bundles have a smaller shifting cost than bundles that were added to the blue successors in Step 2, and thus are the smallest elements in the set.

The data structure used to implement the sets is a 2-3 tree. All the required operations accessing the data structure can be implemented in $O(\log(m+n))$ time, giving us a total running time of $O((n+m)\log(n+m))$.

Theorem 6 *The transportation problem with integer supplies and demands for points on a line can be solved in $O((n+m)\log(n+m))$ time.*

4 Minimum cost matching in bitonic Monge arrays

In this section we describe an $O(n \log m)$ time algorithm for finding a minimum cost perfect matching in bitonic Monge arrays.

4.1 Preliminaries and notations

A matching in an $n \times m$ array $A = \{a[i,j]\}$ is a sequence of n entries, $a[1,j_1], \ldots, a[n,j_n]$, where $j_{i_1} \neq j_{i_2}$ for all $1 \leq i_1 < i_2 \leq n$. A minimum cost matching is a matching where $\sum_{i=1}^{n} a[i,j_i]$ is minimized. The following lemma defines the structure of one of the minimum cost matchings in a Monge array.

Lemma 7 *There exists a minimum cost matching in a Monge array A, such that the entries form a generalized diagonal; that is, for all $i_1 < i_2$, if $a[i_1,j_1]$ and $a[i_2,j_2]$ are two entries in the matching, then $j_1 < j_2$.*

Proof: Consider a minimum cost matching M for A, and suppose that M includes two entries $a[i_1,j_2]$ and $a[i_2,j_1]$ such that $i_1 < i_2$ and $j_1 < j_2$. By the Monge property, $a[i_1,j_1] + a[i_2,j_2] \leq a[i_1,j_2] + a[i_2,j_1]$. Thus, the matching M' obtained by replacing the entries $a[i_1,j_2]$ and $a[i_2,j_1]$ of M by $a[i_1,j_1]$ and $a[i_2,j_2]$ costs at most the same as M. If M' costs less than M, we have a contradiction. Otherwise, M' is a different minimum matching. We can continue in this manner until the resulting matching forms a generalized diagonal. ■

Given Lemma 7, it is easy to see that the case $n = m$ is solved by just taking the main diagonal as the solution. However, the problem is not so simple when $n < m$. The best known algorithm when $n < m$ is a dynamic programming algorithm that runs in time $O(mn)$.

Let A be an $n \times m$ bitonic Monge array. For $1 \leq j \leq m - n + 1$, define $\mathcal{D}_j$ to be the jth (full) diagonal in A; that is, $\mathcal{D}_j$ consists of the entries $a[1,j], a[2,j+1], \ldots, a[n, j+n-1]$. Define $\mathcal{D}_j(i)$ to be the ith element in this diagonal, i.e., $\mathcal{D}_j(i) = a[i, j+i-1]$. Define $\mathcal{D}_j(t,b)$ to be elements of $\mathcal{D}_j$ between rows t and b, i.e., $\mathcal{D}_j(t,b)$ consists of entries $a[t, j+t-1], a[t+1, j+t], \ldots, a[b, j+b-1]$.

4.2 The algorithm

Let A be a bitonic Monge array. From Lemma 7, it follows that there exists a minimum cost matching in A that only contains elements from the set $\{\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_{m-n+1}\}$. Consequently, instead of considering the entire array, we may consider only the set of elements that belong to these diagonals.

Our algorithm uses the "divide and conquer" paradigm. The input to each step of the recursion is an array B of size $(b-t+1) \times (r-\ell+1)$ that contains elements in the set $\{\mathcal{D}_\ell(t,b), \ldots, \mathcal{D}_r(t,b)\}$. Define k to be the index of the middle diagonal of B, $k = \lfloor \frac{r+\ell}{2} \rfloor$. The output is a *separating* row s, $t \leq s \leq b$, that together with the middle diagonal of B, splits B into four quadrants such that there exists a minimum cost matching for B that is inside the top left quadrant and the bottom right quadrant (quadrants II and IV in Figure 4). In other words, there exists a minimum cost matching for B in which rows $t, \ldots, s$ are matched with entries in $B' = \{\mathcal{D}_\ell(t,s), \ldots, \mathcal{D}_k(t,s)\}$, and rows $s+1, \ldots, b$ are matched with entries in $B'' = \{\mathcal{D}_{k+1}(s+1,b), \ldots, \mathcal{D}_r(s+1,b)\}$. Since quadrants II and IV do not share any columns of B it follows that the minimum cost matching for B can be found by two recursive calls: one to an array B' of size $(s-t+1) \times (k-\ell+1)$ and one to an array B'' of size $(b-s) \times (r-k)$.

The algorithm for finding the separating row in B has $(b-t+1)$ stages. In stage $x \geq 1$, we process row $(t-1+x)$ to determine the separating row, s_x, for the subarray given by rows $t, \ldots, t-1+x$. Initially, $s_0 = t-1$ and finally $s = s_{b-t+1}$. In stage $x \geq 1$, the separating row s_x is set either to $t-1+x$ or to s_{x-1}, according to the following criterion. Let $V(i_1,i_2) = \sum_{i=i_1}^{i_2} \mathcal{D}_k(i)$, and let $W(i_1,i_2) = \sum_{i=i_1}^{i_2} \mathcal{D}_{k+1}(i)$.

$$s_x \leftarrow \begin{cases} t-1+x & \text{if } V(s_{x-1}+1, t-1+x) \\ & \quad < W(s_{x-1}+1, t-1+x) \\ s_{x-1} & \text{otherwise} \end{cases}$$

The justification for the definition of s_x is provided in Lemma 10.

We claim that each stage takes constant time. To see this, notice that if $s_{x-1} = t-2+x$, then

$$\begin{aligned} V(s_{x-1}+1, t-1+x) &= \mathcal{D}_k(t-1+x) \\ W(s_{x-1}+1, t-1+x) &= \mathcal{D}_{k+1}(t-1+x) \end{aligned}$$

Otherwise,

$$\begin{aligned} V(s_{x-1}+1, t-1+x) &= V(s_{x-1}+1, t-2+x) + \\ &\qquad \mathcal{D}_k(t-1+x) \\ W(s_{x-1}+1, t-1+x) &= W(s_{x-1}+1, t-2+x) + \\ &\qquad \mathcal{D}_{k+1}(t-1+x). \end{aligned}$$

Hence, it takes constant time to compute each $V(i_1,i_2)$ and $W(i_1,i_2)$, and the separating row can be found in $O(b-t)$ time.

Starting with an array B consisting of diagonals $\mathcal{D}_1, \ldots, \mathcal{D}_{m-n+1}$, we get the following recurrence relation for the running time of our algorithm. Let $T(a,b)$ denote the running time of the algorithm on a matrix with a rows and b diagonals.

$$T(n, m-n+1) = \\ T\left(s, \lfloor \tfrac{m-n}{2} \rfloor + 1\right) + T\left(n-s, \lceil \tfrac{m-n}{2} \rceil\right) + O(n).$$

It is not hard to see that $T(n,1) = O(n)$ since in this case the minimum cost matching is simply the given diagonal. The solution of the recurrence implies the following theorem.

Theorem 8 (Complexity) *The time complexity of the algorithm is $O(n \log m)$.*

The algorithm can be parallelized. The parallel time complexity is $O(\log^2 m)$ time with $n/\log m$ processors on an EREW PRAM.

4.3 Correctness

The correctness proof is based on the following lemma. For $1 \leq \ell < r \leq m-n+1$, consider the sub-array B that consists of the elements in $\{\mathcal{D}_\ell(t,b), \ldots, \mathcal{D}_r(t,b)\}$. Let $U_j = \sum_{i=t}^{b} \mathcal{D}_j(i)$.

Lemma 9 *Suppose that the minimum element in row t is to the right or on $\mathcal{D}_r$ (i.e., the column index of the minimum element in row t is at least r) and that the minimum element in row b is to the left or on $\mathcal{D}_\ell$ (i.e., the column index of the minimum element in row b is at most $\ell + b - 1$). Then, the sequence $U_\ell, \ldots, U_r$ is bitonic.*

Proof: Consider three consecutive diagonals and denote their elements by $x_1, \ldots, x_c$, $y_1, \ldots, y_c$, and $z_1, \ldots, z_c$, respectively, where $c = b-t+1$. Let $X = \sum_{i=1}^{c} x_i$, $Y = \sum_{i=1}^{c} y_i$, and $Z = \sum_{i=1}^{c} z_i$. We claim that $Y - X \leq Z - Y$. Consequently, the sequence of the differences between the diagonals is a monotonic non-decreasing sequences and, therefore, the sequence of diagonals is bitonic.

It remains to prove the above claim. For all i, $1 \leq i \leq c-1$, the Monge property gives: $y_i + y_{i+1} \leq z_i + x_{i+1}$, or, $y_{i+1} - x_{i+1} \leq z_i - y_i$. We can write $Y - X$ and $Z - Y$ as follows:

$$Y-X = (y_1-x_1)+(y_2-x_2)+(y_3-x_3)+\cdots+(y_c-x_c)$$

$$Z-Y = (z_1-y_1)+(z_2-y_2)+\cdots+(z_{c-1}-y_{c-1})+(z_c-y_c)$$

Therefore, $(Y-X)-(Z-Y) \leq (y_1-x_1)-(z_c-y_c)$. Because of the bitonicity of the rows and since the minimum in row t is to the right or on $\mathcal{D}_r$, $y_1 - x_1$ is negative. Similarly, $z_c - y_c$ is positive. Hence, $(Y - X) - (Z - Y) \leq 0$. ■

Now, we prove the correctness of the recursive step. To simplify the notation, we consider the top level of the recursion. Let B_i be the sub-array containing the first i rows of B.

Lemma 10 *For all* $1 \leq i \leq n$, *row* s_i *of* B_i *is a separating row for the sub-array* B_i.

Proof: The proof is by induction on i. For the base case $i = 1$, and s_1 is either 1 or 0, depending on the values of $V(1,1)$ and $W(1,1)$. (By comparing the two values we know where the minima of row 1 is.) Now, assume that row s_i of B_i is a separating row for B_i, and we show that row s_{i+1} of B_{i+1} is a separating row for B_{i+1}.

Case 1: $s_i = i$; that is, there exists an optimal matching for B_i that lies in the first $k = \lfloor \frac{n+m}{2} \rfloor$ diagonals. Denote this matching by M_i.

Case 1.1: $V(s_i+1, i+1) = V(i+1, i+1) < W(s_i+1, i+1) = W(i+1, i+1)$, i.e., the element $\mathcal{D}_k(i+1)$ is less than the element $\mathcal{D}_{k+1}(i+1)$. In this case, the algorithm sets $s_{i+1} = i+1$. To obtain a contradiction suppose that there is no optimal matching for B_{i+1} in diagonals $\mathcal{D}_1, \ldots, \mathcal{D}_k$. Consider an optimal matching for B_{i+1}. By our assumption it must be that the match in row $i+1$ lies in diagonal $\mathcal{D}_z$ for some $z > k$. It can be shown that there exists such an optimal matching M' that contains the matching M_i. However, the matching M_i can be augmented by $\mathcal{D}_k(i+1) = a[i+1, i+k]$, since column $i+k$ does not intersect the first i rows of diagonals $\mathcal{D}_1, \ldots, \mathcal{D}_k$. Since row $i+1$ is bitonic and since $\mathcal{D}_k(i+1) < \mathcal{D}_{k+1}(i+1)$, $\mathcal{D}_k(i+1) < \mathcal{D}_z(i+1)$. Thus, the matching given by adding $\mathcal{D}_k(i+1)$ to M_i is smaller than M'. A contradiction.

Case 1.2: $V(i+1, i+1) \geq W(i+1, i+1)$, i.e., $\mathcal{D}_k(i+1) \geq \mathcal{D}_{k+1}(i+1)$. In this case the algorithm sets $s_{i+1} = i$. From the bitonicity of row $i+1$ it follows that the minimum entry in row $i+1$ lies in one of the diagonals $\mathcal{D}_{k+1}, \ldots, \mathcal{D}_{m-n+1}$. Hence, this minimum element can be added to M_i to form a matching for B_{i+1}, that costs less than any other matching containing M_i. Since it can be shown that there exists an optimal matching for B_{i+1} that contains M_i, $s_{i+1} = i$ is a separating row for B_{i+1}.

Case 2: $s_i < i$, that is, there exists an optimal matching for B_i the first s_i entries of which lie in diagonals $\mathcal{D}_1, \ldots, \mathcal{D}_k$, and the last $i - s_i$ entries of which lie in diagonals $\mathcal{D}_{k+1}, \ldots, \mathcal{D}_{m-n+1}$.

Case 2.1: $V(s_i+1, i+1) < W(s_i+1, i+1)$. In this case the algorithm sets $s_{i+1} = i+1$. Define M' to be the optimal matching for B_{i+1} for which the diagonal index of each row is minimal among all optimal matchings for B_{i+1}. We claim that such an optimal matching always exists and call it the "leftmost" matching. Consider the index of the match of row $i+1$ in M'. If this index is not greater than k, then $s_{i+1} = i+1$ is indeed a separating row.

Suppose this is not the case, and that the match of row $i+1$ lies in diagonal $\mathcal{D}_z$, for $z \geq k+1$. Let $q \leq i+1$ be the minimum index such that the match of row q in M' lies in diagonal $\mathcal{D}_z$. Notice that the minimum element in row q must be to the right or on $\mathcal{D}_z$. Otherwise, $\mathcal{D}_z(q)$ can be substituted by $\mathcal{D}_{z-1}(q)$ to obtain a valid matching that is no worse than M', a contradiction to our assumption that M' is the "leftmost" matching. Since $s_i < i$, we have $V(s_i+1, i) \geq W(s_i+1, i)$. By the assumption $V(s_i+1, i+1) < W(s_i+1, i+1)$. Consequently, the element $\mathcal{D}_k(i+1)$ must be less than the element $\mathcal{D}_{k+1}(i+1)$. This and the bitonicity of the rows imply that the minimum element in row $i+1$ must be to the left or on $\mathcal{D}_k$.

Consider the sub-array given by $\{\mathcal{D}_k(q, i+1), \ldots, \mathcal{D}_z(q, i+1)\}$. This sub-array conforms with the conditions of Lemma 9, implying that the sequence $\sum_{x=q}^{i+1} \mathcal{D}_k(x), \ldots, \sum_{x=q}^{i+1} \mathcal{D}_z(x)$ is bitonic. Since M' is the "leftmost" matching, q must be greater than s_i. Since q has not been chosen as a separating line, $V(s_i+1, q) \geq W(s_i+1, q)$. It follows that $V(q, i+1) < W(q, i+1)$. The bitonicity of the diagonals (Lemma 9) implies that $\sum_{x=q}^{i+1} \mathcal{D}_{z-1}(x) \leq \sum_{x=q}^{i+1} \mathcal{D}_z(x)$. However, in this case the matching given by substituting $\mathcal{D}_z(q), \ldots, \mathcal{D}_z(i+1)$ of M' by the corresponding elements that lie on $\mathcal{D}_{z-1}$ form a valid matching that is no worse than M', a contradiction to our assumption that M' is the "leftmost" matching.

Case 2.2: $V(s_i+1, i+1) \geq W(s_i+1, i+1)$. In this case the algorithm sets $s_{i+1} = s_i$. Let M' be an optimal matching for B_{s_i} that lies in the first k diagonals. Let M'' be the optimal "rightmost" matching for B_{i+1} that contains M'; that is, M'' is the matching that contains M', and in each row $s_i < x \leq i+1$ the diagonal index is maximal among all optimal matchings for B_{i+1}. Consider the index of the match of row s_i+1 in M''. If this index is at least $k+1$, then s_{i+1} is a separating row, and we are done.

Suppose that this is not the case, and that the match of row s_i+1 lies to the left of diagonal $\mathcal{D}_{k+1}$. Let z be the diagonal index of the match of row s_i+1. Let $s_i+1 \leq q \leq i$ be the maximum index such that the match of row q in M'' lies in diagonal $\mathcal{D}_z$. Notice that the minimum element in row q must be to the left or on $\mathcal{D}_z$. Otherwise, $\mathcal{D}_z(q)$ can be substituted by $\mathcal{D}_{z+1}(q)$ to obtain a valid matching that is no worse than M'', a contradiction to our assumption that M'' is the "rightmost" matching. On the other hand, because $V(s_i+1, s_i+1) \geq W(s_i+1, s_i+1)$, and because the rows are bitonic, it follows that the minimum element in row s_i+1 must be to the right or on $\mathcal{D}_{k+1}$.

Consider the sub-array given by $\{\mathcal{D}_z(s_i+1, q), \ldots, \mathcal{D}_{k+1}(s_i+1, q)\}$. This sub-array conforms with the conditions of Lemma 9, implying that the sequence $\sum_{x=s_i+1}^{q} \mathcal{D}_z(x), \ldots, \sum_{x=s_i+1}^{q} \mathcal{D}_{k+1}(x)$ is bitonic. Notice that $V(s_i+1, q) \geq W(s_i+1, q)$. From Lemma 9 we conclude that $\sum_{x=s_i+1}^{q} \mathcal{D}_z(x) \geq \sum_{x=s_i+1}^{q} \mathcal{D}_{z+1}(x)$. However, in this case the matching given by substituting $\mathcal{D}_z(s_i+1), \ldots, \mathcal{D}_z(q)$ of M'' by the corresponding elements that lie on $\mathcal{D}_{z+1}$ form a valid matching that is no worse from M'', a contradiction to our assumption that M'' is the "rightmost" matching.

■

The validity proof of the base recursion is trivial. The correctness of the algorithm follows from substituting $i = n$.

Theorem 11 *The algorithm described in Section 4.2 finds a minimum cost matching in a bitonic Monge array.*

4.4 The transportation problem

Suppose that we are given a transportation problem with integral supplies and demands, and a cost array A that is bitonic Monge. A simple way to transform this problem into a matching problem is by "blowing" the cost array into a $(\sum_{i=1}^{n} b_i) \times (\sum_{j=1}^{m} a_j)$ array, where the (i,j)th entry of the original array is replicated $b_i \times a_j$ times. Applying our algorithm for this array gives an $O((\sum_{i=1}^{n} b_i)\log(\sum_{j=1}^{m} a_j))$ time algorithm. However, the algorithm can be modified such that the amount of work per diagonal is $O(m+n)$, resulting in an $O(m \log(\sum_{j=1}^{m} a_j))$ time algorithm.

We outline the main modifications that are needed to the algorithm. The base case, when we have a single diagonal is quite easy: the matching is a single diagonal. The amount of time it takes to encode this matching is simply the number of *distinct* demand rows and supply columns in this diagonal. The main computational step in the divide and conquer scheme is the algorithm to compute the *separating* row. Suppose that we are computing the separating row for a matrix B (this is the "blown" up matrix) of size $(b-t+1) \times (r-\ell+1)$. If all the rows in B are identical (they correspond to the same demand row in the original cost matrix A), then the min cost matching is a single diagonal, and hence the separating row will either be $t-1$ or b (all the rows will match either to a diagonal $i \leq k$, or a diagonal $i > k$, where $k = \lfloor \frac{r+\ell}{2} \rfloor$). Recall that the diagonals form a bitonic sequence. Thus in time proportional to the number of distinct columns in B, we can figure out which side of k the cheapest diagonal is. When the rows in B are not identical, we only need to compute the value of s_x for a row that is different from its previous row. Hence the algorithm can be made to run in time proportional to the number of distinct rows and columns in B.

Acknowledgments: We would like to thank James Park for telling us about the skiers problem, and for informing us of the paper by Karp and Li [KL75]. The authors also thank Bill Pulleyblank for several useful discussions and for providing useful references.

References

[AM86] A. Aggarwal and R.C. Melville. Fast computation of the modality of polygons. *Journal of Algorithms*, 7: 369–381, 1986.

[AMO89] R.K. Ahuja, T.L. Magnanti, and J. B. Orlin. Network flows. In *Optimization, Handbooks in Operations Research and Management Science*, pages 211–370. North-Holland Publishing, 1989.

[GTT89] A.V. Goldberg, É. Tardos, and R.E. Tarjan. Network flow algorithms. Technical Report TR 860, School of Operations Research and Industrial Engineering, Cornell University, September 1989.

[Hof63] A.J. Hoffman. On simple linear programming problems. In V. Klee, editor, *Convexity: Proceedings of the Seventh Symposium in Pure Mathematics of the AMS*, volume 7, pages 317-327. American Mathematical Society, Providence, RI, 1963.

[KL75] R.M. Karp and S.Y.R. Li. Two Special Cases of the Assignment Problem. *Discrete Mathematics*, 13: 129–142, 1975.

[Kuh55] H.W. Kuhn. The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2(1): 83–97, 1955.

[Lei91] C.E. Leiserson, 1991. Second quiz in the Introduction to Algorithms course, problem Q-3.

[MC80] C. Mead and L. Conway. *Introduction to VLSI systems.* Addison-Wesley, Reading, MA, 1980.

[MS91] O. Marcotte and S. Suri. Fast matching algorithms for points on a polygon. *SIAM Journal on Computing* 20: 405–422, 1991.

[Vai89] P.M. Vaidya. Geometry helps in matching. *SIAM Journal on Computing* 18: 1201-1225, 1989.

[Won91] C.K. Wong, 1991. Personal Communication.

[WPMK86] M. Werman, S. Peleg, R. Melter, and T.Y. Kong. Bipartite graph matching for points on a line or a circle. *Journal of Algorithms*, 7: 277-284, 1986.

[WPR84] M. Werman, S. Peleg, and A. Rosenfeld. A distance metric for multidimensional histograms. Technical Report CAR-TR-90, Center for Automation Research, Univ. of Maryland, August 1984.

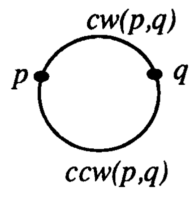

Figure 1: In this figure, $x(p,q) = cw(p,q)$.

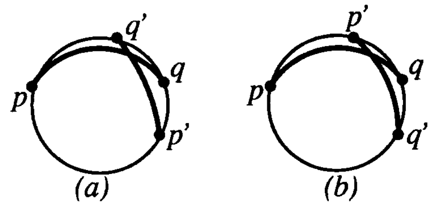

Figure 2: Thicker arcs between points represent the edges of the matching.

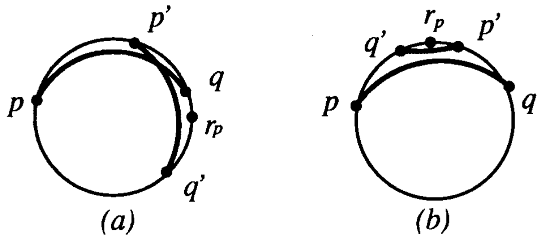

Figure 3: Thicker arcs between points represent the edges of the matching.

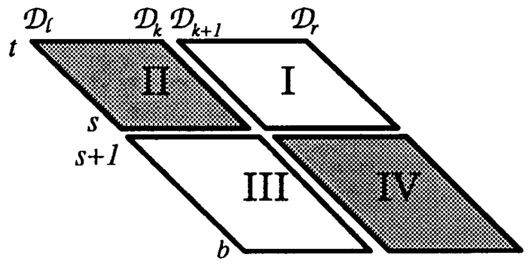

Figure 4: Quadrant *II* includes row s and diagonal k; quadrant IV includes row $s+1$ and diagonal $k+1$.