SIAM J. COMPUT.
Vol. 27, No. 1, pp. 170–201, February 1998

LINEAR AND $O(n \log n)$ TIME MINIMUM-COST MATCHING ALGORITHMS FOR QUASI-CONVEX TOURS*

SAMUEL R. BUSS† AND PETER N. YIANILOS‡

Abstract. Let G be a complete, weighted, undirected, bipartite graph with n red nodes, n' blue nodes, and symmetric cost function $c(x, y)$. A maximum matching for G consists of $\min\{n, n'\}$ edges from distinct red nodes to distinct blue nodes. Our objective is to find a minimum-cost maximum matching, i.e., one for which the sum of the edge costs has minimal value. This is the weighted bipartite matching problem or, as it is sometimes called, the assignment problem.

We report a new and very fast algorithm for an abstract special case of this problem. Our first requirement is that the nodes of the graph are given as a "quasi-convex tour." This means that they are provided circularly ordered as $x_1, \ldots, x_N$, where $N = n + n'$, and that for any x_i, x_j, x_k, x_ℓ, not necessarily adjacent but in tour order, with x_i, x_j of one color and x_k, x_ℓ of the opposite color, the following inequality holds:

$$c(x_i, x_\ell) + c(x_j, x_k) \leq c(x_i, x_k) + c(x_j, x_\ell).$$

If $n = n'$, our algorithm then finds a minimum-cost matching in $O(N \log N)$ time. Given an additional condition of "weak analyticity," the time complexity is reduced to $O(N)$. In both cases only linear space is required. In the special case where the circular ordering is a line-like ordering, these results apply even if $n \neq n'$.

Our algorithm is conceptually elegant, straightforward to implement, and free of large hidden constants. As such we expect that it may be of practical value in several problem areas.

Many natural graphs satisfy the quasi-convexity condition. These include graphs which lie on a line or circle with the canonical tour ordering, and costs given by any concave-down function of arclength — or graphs whose nodes lie on an arbitrary convex planar figure with costs provided by Euclidean distance.

The weak-analyticity condition applies to points lying on a circle with costs given by Euclidean distance, and we thus obtain the first linear-time algorithm for the minimum-cost matching problem in this setting (and also where costs are given by the L_1 or L_∞ metrics).

Given two symbol strings over the same alphabet, we may imagine one to be red and the other blue and use our algorithms to compute string distances. In this formulation, the strings are embedded in the real line and multiple independent assignment problems are solved, one for each distinct alphabet symbol.

While these examples are somewhat geometrical, it is important to remember that our conditions are purely abstract; hence, our algorithms may find application to problems in which no direct connection to geometry is evident.

Key words. assignment problem, bipartite weighted matching, computational geometry, concave penalty function, convexity, linear time, Monge property, quadrangle inequality, string comparison

AMS subject classifications. 05C70, 05C85, 05C90, 52A37, 68Q20, 68R10, 68U15, 90C27

PII. S0097539794267243

1. Introduction.

The above abstract gives a short overview of the contents of the paper, and we shall give an in-depth discussion of our definitions, results, and algorithm below. However, we first give a quick review of prior related work on

*Received by the editors May 9, 1994; accepted for publication (in revised form) January 7, 1996. A preliminary version of this paper appeared in the *Proceedings of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, Arlington, VA, SIAM, Philadelphia, 1994, pp. 65–76.

http://www.siam.org/journals/sicomp/27-1/26724.html

†Department of Mathematics, University of California, San Diego, La Jolla, CA 92092-0112 (sbuss@ucsd.edu). The research of this author was supported in part by NSF grants DMS-9205181 and DMS-9503247.

‡NEC Research Institute, 4 Independence Way, Princeton, NJ 08540 and Department of Computer Science, Princeton University, 35 Olden Street, Princeton, NJ 08544-2087 (pny@research.nj.nec.com).

matching. We shall consider graphs G which have N nodes; the nodes are partitioned into a set of n red nodes and n' blue nodes with $N = n + n'$. G is *balanced* if it has equal numbers of red and blue nodes. There is a symmetric cost function $c(x, y)$, which gives the cost of an edge from node x to node y, with x and y of distinct colors. A matching is a set of edges with no endpoints in common that match all the nodes of one color with the same number of nodes of the opposite color. The cost of a matching is the sum of the costs of its edges. The problem of finding a minimal-cost matching for a general bipartite graph is known to have an $O(N^3)$ time algorithm (see Lawler [18] for this and other background on matching), and for graphs with nodes in the plane with the Euclidean distance as cost function, there is a $O(N^{2.5} \log N)$ time algorithm due to Vaidya [22].

The minimum-cost matching problem is substantially easier in the case where the nodes are in line-like order or are circularly ordered. The simplest versions of line-like/circular orderings are where the points lie on a line or lie on a curve homeomorphic to a circle, and the cost $c(x, y)$ of an edge between x and y is equal to the shortest arclength distance between the nodes. The matching problem for this arclength cost function has been studied by Karp and Li [14], Aggarwal et al. [1], Werman et al. [23], and others, and is the "skis and skiers" problem of Lawler [18]. Karp and Li have given linear time algorithms for this matching problem; Aggarwal et al. have generalized the linear time algorithm to the transportation problem.

A more general version of the matching problem for graphs in line-like order has been studied by Gilmore and Gomory [10] (see [18]). In this version, the cost of an edge from a red node x forward to a blue node y is defined to equal $\int_x^y f$, and from a blue node x forward to a red node y to equal $\int_x^y g$, for some functions f and g. This matching problem has a linear time algorithm provided $f + g \geq 0$.

Another version of the matching problem for line-like graphs is considered by Aggarwal et al. [1]; they use graphs which satisfy a "Monge" property which states that the inequality (1.1) below holds except with the inequality sign's direction reversed. They give a linear time algorithm for the matching problem for (unbalanced) Monge graphs.

In the prior work most closely related to this paper, Marcotte and Suri [20] consider the matching problem for a circularly ordered, balanced tour in which the nodes are the vertices of a convex polygon and the cost function is equal to Euclidean distance. This matching problem is substantially more complicated than the comparatively simple "skis and skiers" type problems; nonetheless, Marcotte and Suri give an $O(N \log N)$ time algorithm which solves this minimum-cost matching problem. For the case where the nodes are the vertices of a simple polygon and the cost function is equal to the shortest Euclidean distance *inside* the polygon, they give an $O(N \log^2 N)$ time algorithm.

The main results of this paper apply to all of the above matching problems on circularly ordered or line-like tours, with the sole exception of unbalanced, Monge graphs. For the "skis and skiers" and the problems of Gilmore and Gomory, Theorem 1.9 gives new linear time algorithms that find minimum-cost matchings which are different than the traditional minimum-cost matchings (and our algorithms are more complicated than is necessary for these simple problems). Our algorithms subsume those of Marcotte and Suri and give some substantial improvements. First, with the weak analyticity condition, we have linear time algorithms for many important cases, whereas Marcotte and Suri's algorithm takes $O(N \log N)$ time. Second, our assumption of quasi convexity is considerably more general than their planar geometrical setting and allows diverse applications. Third, our algorithms are conceptually simpler than the divide-and-conquer methods used by Marcotte and Suri, and we expect that our algorithms are easier to implement.

All of our algorithms have been implemented as reported in [5]; a brief overview of this implementation is given in section 3.4.

We list some sample applications of our algorithms in the examples numbered 1–8 below. One example of a matching problem solution is shown in Figure 1.1. For this figure, a 74-node bipartite graph was chosen with nodes on the unit circle. For this matching problem, the cost of an edge is equal to the Euclidean distance between its endpoints. The edges shown form a minimum-cost matching.

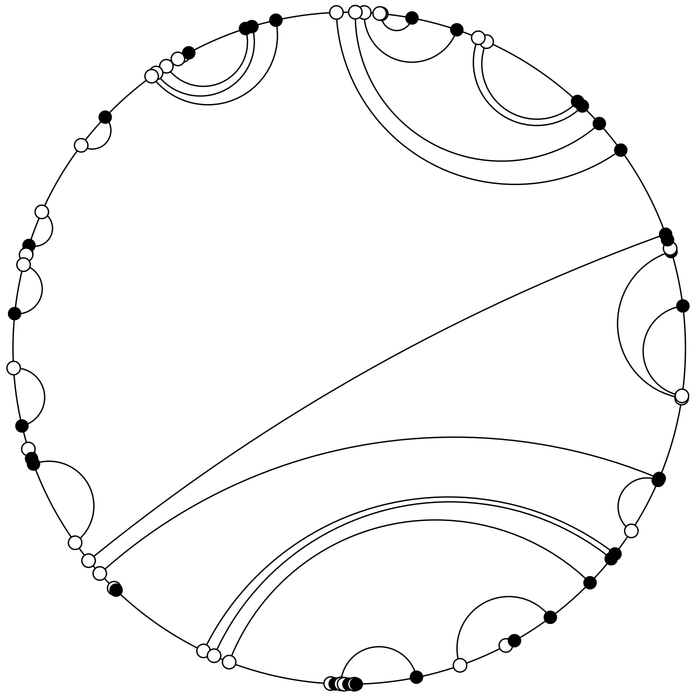

FIG. 1.1. *The minimum-cost matching for a 74-node graph on the circle with Euclidean distance as the cost function.*

Our quasi-convex property is equivalent to the "inverse quadrangle inequality" used, for instance, by [8] but is weaker than the similar "inverse Monge property" of [4]. In fact, we show below that any Monge matching problem may be trivially transformed into a quasi-convex matching problem, but not vice-versa.

Dynamic programming problems based on cost functions which satisfy the (inverse) quadrangle inequality, and some closely related matrix-search problems, have been studied by many authors including [2, 3, 4, 7, 8, 9, 12, 15, 16, 17, 19, 24, 25]. However, we have discovered no direct connection between our quasi-convex matching problem and the problems solved by these authors.

The notion of a *Monge array* [13] is related to that of quasi convexity, but the Monge condition is stronger (i.e., quasi convexity is strictly more general). Because of the similarity between the definitions of both properties, we take the time to illustrate this point in detail. To understand the Monge property in our bipartite setting, imagine the cost function to be an array, and impose the restriction that its first argument select a red point and the second a blue point. The array is Monge provided that for all i, j, k, ℓ satisfying $1 \le i < j \le n$ and $1 \le k < \ell \le n'$, we have

$$c(R_i, B_k) + c(R_j, B_\ell) \le c(R_i, B_\ell) + c(R_j, B_k).$$

Now given a graph with a Monge cost array, we convert it (in linear time) to a quasi-convex tour by simply visiting the red vertices first, in Monge order, followed by the blue vertices, in reverse Monge order. The quasi-convexity inequality is then an immediate consequence of the Monge property and our reverse ordering of the blue vertices. This reversal is necessary because the sense of the Monge inequality is opposite that of quasi convexity.

However, not every quasi-convex tour can be rearranged to form a Monge array. We will now exhibit such a quasi-convex tour. Its nodes lie along the real line, and costs are given by the square root of internode distance; tour order is from left to right. Given a subtour of the form $R_iR_jB_aB_b$, then it is easily shown that in any Monge reordering, $B_a \prec B_b$ iff $R_j \prec R_i$. Similarly, given a tour of the form $R_jB_aB_bR_i$, $B_a \prec B_b$ iff $R_j \prec R_i$. Our counterexample then consists of any tour having a subtour of the form $RBBRRBB$. To understand why, we attach subscripts resulting in $R_iB_aB_bR_jR_kB_cB_d$ and proceed to apply the two rules above to get the implications

$$B_a \prec B_b \implies R_i \prec R_j \implies B_d \prec B_c \implies R_j \prec R_k.$$

Also,

$$B_a \prec B_b \implies R_k \prec R_j,$$

which is a contradiction. Symmetrically the same conclusion is reached if one begins instead with $B_b \prec B_a$, whence no Monge rearrangement exists.

We now give the definitions necessary to state the main results of this paper. We think of the nodes of the graph G as being either a line-like or circular tour of the graph; in the case of a circular tour, we think of the node x_1 as following again after x_N.

DEFINITION 1.1. *A sequence of nodes $x_{i_1}, x_{i_2}, \ldots, x_{i_\ell}$ are* in input order *if and only if $i_1 < i_2 < \cdots < i_\ell$. The nodes are defined to be* in tour order *if and only if there exists a k such that the sequence $x_{i_k}, \ldots, x_{i_\ell}, x_{i_1}, \ldots, x_{i_{k-1}}$ is in input order.*

DEFINITION 1.2. *The nodes $x_1, \ldots, x_N$ form a* quasi-convex *tour if and only if, whenever x_i, x_j, x_k, x_ℓ are in tour order, with x_i and x_j of one color and x_k and x_ℓ of the other color, then*

$$c(x_i, x_\ell) - c(x_i, x_k) \le c(x_j, x_\ell) - c(x_j, x_k). \tag{1.1}$$

Reordering terms in (1.1) gives

$$c(x_i, x_\ell) + c(x_j, x_k) \ \le\ c(x_i, x_k) + c(x_j, x_\ell).$$

To give a geometric intuition to quasi convexity, note that when x_i, x_j, x_k, x_ℓ are the vertices of a quadrilateral, the inequality states that the sum of the lengths of diagonals is greater than or equal to the sum of the lengths of two of the sides.

DEFINITION 1.3. *The tour $x_1, \ldots, x_N$ of G is* line-like *if and only if the following holds: For all $i < j < k$, we have*

$$c(x_i, x_j) \le c(x_i, x_k)$$

if x_i is of opposite color from x_j and x_k, and we have

$$c(x_i, x_k) \geq c(x_j, x_k)$$

if x_k is of opposite color from x_i and x_j.

The property of quasi convexity is defined independently of the starting point of the tour; i.e., the nodes of the tour can be "rotated" without affecting quasi convexity. Obviously, the definition of line-like tours is sensitive to the choice of starting point of the tour.

Our main theorems give either $O(N \log N)$ or $O(N)$ time algorithms for all of the following examples, with the exception of example 7:

1. Let the nodes $x_1, \ldots, x_N$ be sequentially ordered points on a line (e.g., they are real numbers indicating points on the x-axis), and let $||x_j - x_i||$ be the Euclidean distance from x_i to x_j. Let f be any concave-down function, so $f''(x) \leq 0$ for all x. If the cost function is defined by

$$c(x_i, x_j) \;=\; f(||x_j - x_i||), \tag{1.2}$$

then $x_1, \ldots, x_N$ form a quasi-convex tour. Prior work for examples 1 and 2 gave linear time matching algorithms *only* for the case where $f(x)$ is a linear function [14, 1].

2. Now let the points $x_1, \ldots, x_N$ lie on a smooth curve C which is homeomorphic to a circle, with the points listed in, say, counterclockwise order. And let $||x_j - x_i||$ equal the shortest arclength along C from x_i to x_j. Again let $f(x)$ be any concave down function. With the cost function given by equation (1.2), the nodes $x_1, \ldots, x_N$ form a quasi-convex tour.

3. Suppose $x_1, \ldots, x_N$ lie, in that order, on a circle. Let $c(x_i, x_j)$ equal the Euclidean distance from x_i to x_j. Since Euclidean distance is a concave-down function of the circular arclength, this is a special case of example 2 and the nodes form a quasi-convex tour. In this case, the weak analyticity condition always holds and Main Theorem 1.9 gives an $O(N)$ time algorithm. The best prior algorithm was $O(N \log N)$ time [20].

4. More generally, if $x_1, \ldots, x_N$ are the vertices of a convex polygon listed in, say, counterclockwise order, and if the cost function is equal to Euclidean distance, then the nodes form a quasi-convex tour. The prior algorithm for this case was $O(N \log N)$ time [20] and our algorithms are either $O(N)$ or $O(N \log N)$ time depending on whether the weak analyticity condition holds.

5. Some nonconvex polygons also have vertices which form a quasi-convex tour. For example, in a polygon shaped as in Figure 1.2, the vertices A, B, C, D will form a quasi-convex tour, provided the angle θ not too large. (This is why we use "quasi-convex" instead of "convex" to describe tours which satisfy equation (1.1).)

6. Examples 4 and 5 are also quasi-convex under other distance metrics such as the L_1 and L_∞ metrics.

7. Marcotte and Suri consider graphs where the nodes are the vertices of a simple polygon and the cost function is equal to the length of the shortest connecting path inside the polygon. The nodes of such a polygon form a quasi-convex tour. The prior algorithm and the algorithm of this paper are $O(N \log^2 N)$ time for this example, since the cost function requires $O(\log N)$ time to compute.

8. In string matching algorithms, the cost of shifting a character's position is specified as a function of the distance shifted. The authors have worked in the past on string matching algorithms [26, 27] in which the cost function is a linear function

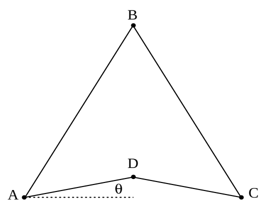

FIG. 1.2. *A quasi-convex polygon which is not geometrically convex.*

of distance. These prior algorithms have been quite successfully used in commercial applications, especially natural language search, and we expect that the use of a concave-down distance function will significantly improve the matching quality. As we discuss in section 5, the setting of example 1 above is precisely what is needed to allow (near) linear time string matching algorithms with concave-down cost functions. A number of authors, including [7, 8], have studied concave-down cost functions for string matching; their string matching algorithms are based on least-edit-distance and, in this regard, are quite different from ours. Least-edit-distance string matching algorithms are widely used because they provide rich and flexible string comparison functions; on the other hand, the best general algorithms for computing least-edit-distance require $O(N^2)$ time (see [21]). Our string matching algorithms are not as flexible but can be tailored to work well for many applications; they have the advantage of being linear time computable.

MAIN THEOREM 1.4.

(i) *There is an $O(N \log N)$ time algorithm for the minimum-cost matching problem for line-like quasi-convex tours.*

(ii) *There is an $O(N \log N)$ time algorithm for the minimum-cost matching problem for balanced quasi-convex tours.*

Remark. The running times of the algorithms are given in terms of the number N of nodes, even though the input size may in some cases need to be $\Omega(N^2)$ to fully specify the values of the cost function. However, in all the examples above, the input size is $O(N)$ since the cost function is specified by the nodes' positions on a line, on a curve, or in the plane. In any event, our runtime analysis assumes that any value $c(x_i, x_j)$ of the cost function can be computed in constant time. If this is not the case, then the runtimes are to be multiplied by the time needed to compute a value of the cost function; this is the situation in example 7 above.

We next define a "weak analyticity" condition which will allow even faster algorithms.

DEFINITION 1.5. *Suppose that x_i and x_j are red (blue) nodes, that $\delta \geq 0$, and that there is a blue (respectively, red) node x_k such that*

$$c(x_i, x_k) - c(x_j, x_k) < \delta.$$

The δ-crossover point *of x_i and x_j is defined to be the first such x_k, where "first" means in tour order starting from x_j and ending at x_i. If no such x_k exists, then the δ-crossover point does not exist.*

It is not hard to see that the property of quasi convexity implies that, if the δ-crossover point x_k exists, then $c(x_i, x_\ell) - c(x_j, x_\ell) \geq \delta$ whenever x_i, x_j, x_ℓ, x_k are in tour order and $c(x_i, x_\ell) - c(x_j, x_\ell) < \delta$ whenever x_i, x_j, x_k, x_ℓ are in tour order. Thus binary search provides an $O(\log N)$ time procedure which, given x_i, x_j, and δ, will determine if x_k exists and, if so, which node is x_k. This is the approach taken in the algorithms of Theorem 1.4 and is the source of the $\log N$ factor in the runtime. However, in some cases, x_k can be found in constant time and we define the following.

DEFINITION 1.6. *A quasi-convex tour satisfies the* strong analyticity condition *provided there is a constant-time algorithm which can determine if the δ-crossover point of x_i and x_j exists and, if so, can determine which node it is.*

A quasi-convex tour satisfies the analyticity condition *provided there is a constant-time algorithm which can answer the following question (as a function of similarly colored nodes x_i, x_j, x_k in tour order and of $\epsilon, \delta > 0$, where the δ-crossover of x_i and x_j is known to exist):*

> *"Do x_j and x_k have an ϵ-crossover point which either equals or precedes in tour order the δ-crossover point of x_i and x_j?"*

Even the analyticity condition is too strong to be satisfied in many situations, so we also define a "weak analyticity condition" as follows.

DEFINITION 1.7. *Let x be a node and y and z be denotations of nodes. We write $y \prec_x z$ to denote that either* (i) *y and z exist and are distinct and y precedes z in the tour order beginning at x, or* (ii) *y exists and z does not.*

A relative crossover procedure *is a procedure Ω such that, given ϵ, δ, x_i, x_j, and x_k as input, and letting y be the δ-crossover of x_i and x_j, and z be the ϵ-crossover of x_j and x_k, then*

(i) *If $y \prec_{x_j} z$, then Ω outputs "Yes."*
(ii) *If $z \prec_{x_j} y$, then Ω outputs "No."*
(iii) *Otherwise Ω may output either answer.*

Note that Ω is not required to determine y and z. The difference between weak analyticity and ordinary analyticity is that when condition (iii) *holds, Ω may output either answer.*

DEFINITION 1.8. *The* weak analyticity condition *is said to hold provided there is a constant-time relative crossover procedure.*

Clearly the strong analyticity condition implies the analyticity condition, which in turn implies the weak analyticity condition. In most applications, we do not have the analyticity or strong analyticity conditions, but the weak analyticity condition does hold in many natural situations. In particular, examples 1, 2, 3, and 4 do satisfy the weak analyticity condition provided that the concave-down function is sufficiently natural. Consider, for instance, example 1 with the concave-down function $f(x) = x$, $f(x) = \sqrt{x}$, or $f(x) = \log x$, etc. For example 1, the input nodes $x_1, \ldots, x_N$ are given with a sequence of real numbers $r_1 \le r_2 \le \cdots \le r_N$ which are the positions of the nodes on the real line. Given nodes x_i, x_j and $\delta > 0$, the first possible position for the δ-crossover of x_i and x_j can be found by solving the equation $f(y-r_i) = \delta + f(y-r_j)$ for y; since we assume that arithmetic operations take constant time, the solution y can be found in constant time. Note that y is only the *theoretical* crossover point; the actual crossover is the first node x_k such that $y \le r_k$. Unfortunately, even after y is known, it will not be possible to determine x_k in constant time unless some additional information is given about the distribution of the nodes on the real line. Thus, the analyticity condition and strong analyticity conditions do not hold in general for example 1. The reason the analyticity condition does not hold is that, if the theoretical ϵ-crossover point occurs after the theoretical δ-crossover point, then the analyticity algorithm must output "No" if there is a node after the theoretical δ-crossover point and before or at the theoretical ϵ-crossover point, and must output "Yes" otherwise (because in the latter case the two actual crossover points coincide). Unfortunately, there is no general way to decide this in constant time, so the analyticity condition is false. However, the weak analyticity condition does hold, since the function Ω may operate by computing the theoretical δ-crossover of x_i and x_j and the theoretical ϵ-crossover of x_j and x_k and outputting "Yes" if the former is less than the latter.

For similar reasons, example 3 satisfies the weak analyticity condition; in this case, since the nodes lie on a circle and the cost function is Euclidean distance, the theoretical crossover position is computed (in constant time) as the intersection of a hyperbola and the circle. Likewise, the weak analyticity condition also holds for example 2 if the concave-down function is sufficiently nice, and it holds for example 6, where nodes lie on a circle under the L_1 and L_∞ metrics. Example 4, where the nodes form the vertices of a convex polygon, does not seem to satisfy the weak analyticity condition in general; however, some important special cases do. For example, if the vertices of the convex polygon are known to lie on a polygon with a bounded number of sides, on an oval, or on a branch of a hyperbola, then the weak analyticity condition does hold.

The analyticity condition has been implicitly used by Hirschberg and Larmore [12] who defined a *Bridge* function which is similar to our Ω function. They give a special case in which *Bridge* is constant-time computable and thus the analyticity condition holds. Later, Galil and Giancarlo [8] defined a "closest zero property" which is equivalent to our strong analyticity condition.[1] As we illustrated above, the analyticity and strong analyticity conditions rarely hold. Thus it is interesting to note that the algorithms of Hirschberg and Larmore and of Galil and Giancarlo will still work, with only minor modifications, if only the weak analyticity condition holds.

Our second main theorem implies that these examples which satisfy the weak analyticity condition have linear time algorithms for minimum-cost matching.

MAIN THEOREM 1.9.

(i) *There is an $O(N)$ time algorithm for the minimum-cost matching problem for line-like quasi-convex tours which satisfy the weak analyticity condition.*

(ii) *There is an $O(N)$ time algorithm for the minimum-cost matching problem for balanced quasi-convex tours which satisfy the weak analyticity condition.*

Remark. In order to achieve the linear time algorithms, it is necessary that nodes of the graph be input in their tour order. This assumption is necessary, since without it, it is possible to give a linear time reduction of sorting to the matching problem for line-like tours.

Our main theorems also apply to minimum-cost matchings for some nonbipartite quasi-convex tours. If a nonbipartite graph G has N nodes and cost function c, then a matching for G is a set of $\lfloor \frac{1}{2}N \rfloor$ edges with all endpoints distinct. Part (i) of Main Theorems 1.4 and 1.9 hold also for nonbipartite graphs which are line-like quasi-convex tours. And part (ii) of Main Theorems 1.4 and 1.9 hold also for nonbipartite graphs which are quasi-convex tours with an even number of nodes. The nonbipartite cases are discussed in section 4; the algorithms are simple modifications of the algorithms for the bipartite tours.

It is apparent that our algorithms can be parallelized, but we have not investigated the precise runtime and processor count that is needed for a parallel implementation. He [11] has given a PRAM implementation of Marcotte and Suri's algorithm which uses N processors and $O(\log^2 N)$ time and it is clear that our algorithm can be

[1]The definition of the "closest zero property" is misstated in [8]; it should be defined as saying that it is possible to find the first r such that $w(l,r) - w(k,r) - a \leq 0$ (note their w corresponds to our cost function c, and a is a real). However, their algorithm explicitly uses the correct definition of "closest zero property" (see their Fact 2).

computed with the same number of processors with the same time bounds using He's methods.

Our algorithms apply to unbalanced tours only if they are line-like. This is because in the line-like case the leveling process, described in the next section, induced by choosing the first node as starting point, is guaranteed to decompose the problem into alternating color subproblems, which may be independently solved and reassembled to produce an overall solution. Now, some of these subproblems may be unbalanced, but again using the line-like property, we are able to force balance by adding a dummy node when necessary. These then are two different uses of the line-like property.

In balanced, unimodal tours[2] such as the circle, the leveling concept of section 2 holds in a weaker form. However, we have been unable to extend our results to the unbalanced unimodal case. As an example of the difficulty of this, consider the highly eccentric ellipse of Figure 1.3; the bipartite tour containing its four nodes is unbalanced and is neither line-like nor unimodal. Notice that no starting point induces a leveling which places R_2 and B_1 at the same level, despite the fact that the minimum-cost matching consists of an edge between them. The path to extending our methods to such cases is therefore less clear.

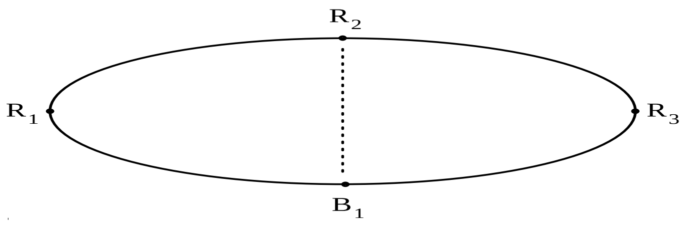

FIG. 1.3. *A bipartite, unbalanced, unimodal tour for which no leveling process works.*

2. Reductions and lemmas.

2.1. Reduction to tours of alternating colors. The first step to giving our minimum-cost matching algorithms is to reduce to the special case of tours in which the colors of the nodes alternate. In other words, we will be able to assume w.l.o.g. that $x_1, x_3, x_5, \ldots$, are red and that $x_2, x_4, x_6, \ldots$, are blue.

DEFINITION 2.1. *Let x_i and x_j be nodes. We write $[x_i, x_j]$ to denote the sequence of nodes obtained by starting with x_i and advancing in tour order to x_j. We write $(x_i, x_j]$, $[x_i, x_j)$, and (x_i, x_j) for this sequence minus the starting node, the ending node, or both.*

If x is a node, let $d(x)$ denote the number of red nodes in $[x_1, x)$ minus the number of blue nodes in $[x_1, x)$. The level *of x, level(x), is equal to $d(x)$ if x is blue and is equal to $d(x) + 1$ if x is red. We write $x \sim y$ to mean that level(x) = level(y); obviously, $\sim$ is an equivalence relation. It is easy to see that if y is the first node after x in input order such that $x \sim y$, then x and y are of opposite colors. Also, if $x \sim y$ and x, y are in input order and are of opposite colors, then (x, y) contains equal number of red and blue nodes. For balanced tours, the $\sim$-equivalence relation is invariant under circular rotation of the nodes in the tour.*

Given a matching on the nodes of a graph, we write $x_i \leftrightarrow x_j$ to indicate the presence of an edge between x_i and x_j in the matching. We say that x_j immediately

[2]In unimodal tours, the cost function from any node rises and then falls as the tour is traversed.

follows x_i *in tour order if* $j = i + 1$ *or if* $i = N$ *and* $j = 1$*. Two nodes* x_i *and* x_j *are* adjacent *if and only if one of them immediately follows the other. An edge* $x_i \leftrightarrow x_j$ *is called a* jumper *if* x_i *and* x_j *are not adjacent. Two jumpers are said to* cross *if they are of the form* $x_i \leftrightarrow x_k$ *and* $x_j \leftrightarrow x_\ell$ *with* x_i, x_j, x_k, x_ℓ *in tour order.*

LEMMA 2.2. *Let* G *be either a line-like quasi-convex tour or a balanced quasi-convex tour. Then* G *has a minimum-cost matching in which every edge* $x_i \leftrightarrow x_j$ *satisfies* $x_i \sim x_j$*. In other words, some minimum-cost matching for* G *can be obtained as a union of minimum-cost matchings on the* $\sim$*-equivalence classes of* G*.*

To prove Lemma 2.2 we use the following lemma.

LEMMA 2.3. G *has a minimum-cost matching in which no jumpers cross.*

Sketch of proof. If a minimum-cost matching does have a pair of jumpers which cross, the quasi-convexity property allows them to be "uncrossed" without increasing the total cost. Repeatedly uncrossing jumpers will eventually yield a minimum-cost matching with no crossing jumpers. (See Lemma 1 of [1] for a detailed proof of this.)

Lemma 2.2 is proved by noting that a minimum-cost matching with no crossing jumpers must respect the $\sim$-equivalence classes. This is because, if a jumper $x_i \leftrightarrow x_j$ is in a crossing-free matching with $i < j$, then the nodes in the interval (x_i, x_j) must be matched with each other and thus (x_i, x_j) must have equal numbers of red and blue nodes. In the unbalanced, line-like case, this also depends on the fact that, w.l.o.g., there is no jumper which crosses an unmatched node (this is an immediate consequence of the line-like condition). □

By Lemma 2.2, in order to find a minimum-cost matching, it suffices to extract the $\sim$-equivalence classes and find minimum-cost matchings for each equivalence class independently. It is an easy matter to extract the $\sim$-equivalence classes in linear time by using straightforward counting. Each equivalence class consists of an alternating color subtour; in the balanced case, there are an even number of nodes in each equivalence class, and in the line-like condition case, there may be an even or odd number of nodes. Thus, to give (near) linear time algorithms for finding matchings, it will suffice to restrict our attention to tours in which the nodes are of alternating colors.

In view of Lemma 2.3, we may restrict our attention to matchings which contain no crossing jumpers. Such a matching will be called *crossing-free*.

Finally, we can assume w.l.o.g. that the tour is balanced. To see why we can assume this, suppose that $x_1, \ldots, x_N$ is an unbalanced, line-like tour of alternating colors. This means that x_1 and x_N are the same color, say red. We can add a new node x_{N+1} to the end of the tour, label it blue, and let $c(x_i, x_{N+1}) = 0$ for all red x_i. These $N + 1$ nodes no longer form a line-like tour; however, they do form a balanced quasi-convex tour. Solving the matching problem for the $N + 1$ nodes immediately gives a solution to the matching problem on the original N nodes.

2.2. Some important lemmas.

Since we are now working only with balanced quasi-convex tours of alternating colors, we shall often change the names of the nodes to $R_1, B_1, \ldots, R_M, B_M$; so R_i and B_j refer to the ith red node and the jth blue node in the tour, respectively. (So x_{2i-1} is the same as R_i and x_{2i} is the same as B_i.) Note that this means $N = 2M$. To simplify notation, we define

$$c_i \;=\; c(R_i, B_i) \qquad \text{and} \qquad c'_i \;=\; c(B_i, R_{i+1}).$$

A *greedy* matching is a matching which contains no jumpers, i.e., every node is matched to an adjacent node. There are two greedy matchings, namely, the one containing all edges $R_i \leftrightarrow B_i$ and the one containing all edges $B_{i-1} \leftrightarrow R_i$ and the edge $B_M \leftrightarrow R_1$. For x_i and x_j nodes of opposite color, a matching σ is said to be *greedy on* (x_i, x_j) provided it contains as a submatching the unique matching of adjacent nodes contained in the interval (x_i, x_j). We similarly define the notion of σ being greedy on a balanced interval I, where I is one of the intervals $[x_i, x_j)$, $[x_i, x_j]$, or $(x_i, x_j]$, but with the additional provisos that $x_i \leftrightarrow x_{i+1}$ is in σ in the first two cases and that $x_{j-1} \leftrightarrow x_j$ is in σ in the second two cases.[3]

The notation $[R_i, B_j]$ has already been defined. In addition, the notation $[i, j]$ denotes the interval of integers $i, i+1, \ldots, j$ if $i < j$, or the (circular) interval $i, i+1, \ldots, M, 1, 2, \ldots, j$ if $j < i \le M$. We also use the notations $(i, j]$, $[i, j)$, and (i, j) for the intervals with one or both of the endpoints omitted.

DEFINITION 2.4. *Let R_i and B_j be nodes; we write $R_i \to B_j$ to denote a* directed *edge going from R_i forward (in tour order) to B_j. That is, we think of $R_i \to B_j$ jumping over the nodes $R_i, B_i, R_{i+1}, \ldots, R_j, B_j$. We say that $R_i \to B_j$ is a* candidate *(meaning, a candidate for a jumper), if*

$$c(R_i, B_j) + \sum_{\ell \in [i,j)} c'_\ell \; < \; \sum_{\ell \in [i,j]} c_\ell .$$

The intuitive meaning of $R_i \to B_j$ being a candidate is that it would be of lower cost to use the jumper $R_i \leftrightarrow B_j$, plus the greedy matching of adjacent nodes in (R_i, B_j), in place of just the greedy matching of adjacent nodes in $[R_i, B_j]$.

A similar definition is used to define what it means for an edge $B_i \to R_j$ to be a candidate; namely, $B_i \to R_j$ is a candidate if and only if

$$c(B_i, R_j) + \sum_{\ell \in (i,j)} c_i \; < \; \sum_{\ell \in [i,j)} c'_i .$$

Candidates always have endpoints of opposite colors and are directed. It is possible to have both $R_i \to B_j$ and $B_j \to R_i$ be (distinct) candidates or to have one or neither of them candidates.

It is an easy observation that if there are no candidates, then the greedy assignment(s) are minimum-cost matchings. To prove this, suppose σ is a minimum-cost matching which contains a jumper; by Lemma 2.3, σ may be picked to contain no crossing jumpers. Since there are no crossing jumpers, σ must contain a jumper $x_i \leftrightarrow x_j$ such that σ is greedy on (x_i, x_j) (namely, pick the jumper so as to minimize the tour-order distance from x_i to x_j). Let σ' be the matching which is the same as σ except greedy on $[x_i, x_j]$. Clearly σ' has one fewer jumper than σ, and since $x_i \to x_j$ is not a candidate, σ' has cost no greater than σ. Iterating this construction shows that at least one of the jumperless greedy matchings must be minimum-cost. To show they are both minimum-cost, let σ_0 and σ_1 be the greedy matchings which contain the edges $x_1 \leftrightarrow x_2$ and $x_1 \leftrightarrow x_N$, respectively. Then σ_0 cannot have cost lower than (respectively, higher than) the cost of σ_1 since otherwise, $x_2 \to x_1$ ($x_1 \to x_N$, respectively) would be a candidate.

DEFINITION 2.5. *A candidate $x_i \to x_j$ is a* minimal *candidate if and only if there is no other candidate $x_k \to x_\ell$ in its interior; that is to say, there is no candidate $x_k \to x_\ell$ with $[x_k, x_\ell]$ a proper subset of $[x_i, x_j]$.*

LEMMA 2.6. *Consider a balanced quasi-convex tour of alternating colors.*

[3]Note that, of the two greedy matchings for G, one is greedy on $[x_1, x_N]$ and the other is greedy on $[x_2, x_1]$.

(i) *Suppose $R_a \to B_b$ is a minimal candidate. Then every minimum-cost, crossing-free matching is greedy on the interval (R_a, B_b). That is to say, every minimum-cost, crossing-free matching contains the edges $B_{\ell-1} \leftrightarrow R_\ell$ for all $\ell \in (a, b]$.*

(ii) *Suppose $B_a \to R_b$ is a minimal candidate. Then every minimum-cost, crossing-free matching is greedy on the interval (B_a, R_b). That is to say, every minimum-cost, crossing-free matching contains the edges $R_\ell \leftrightarrow B_\ell$ for all $\ell \in (a, b)$.*

Note that Lemma 2.6 says only that the edges connecting adjacent nodes in the *interior* of the minimal candidate are in every minimum-cost matching; it does not say that the minimal candidate itself is a jumper in any minimum-cost matching. The proof of Lemma 2.6 is fairly involved and we postpone it until section 2.3. Lemma 2.6 also holds for line-like tours with alternating colors for candidates $x \to y$ with x, y in input order.

Lemma 2.6 suggests an algorithm for finding a minimum-cost matching. Namely, if there is a minimal candidate, greedily assign edges in its interior according to Lemma 2.6. This induces a matching problem on the remaining unassigned nodes, and it is clear that any minimum-cost matching on this smaller problem will lead to a minimum-cost matching for the original problem. Iterating this, one can continue removing nodes in the interiors of minimal candidates and reducing the problem size. Eventually a matching problem with no candidates will be reached; in this case, it suffices to greedily match the remaining nodes.

Unfortunately, this algorithm suggested by Lemma 2.6 is not linear time (yet); thus we need to refine Lemma 2.6 somewhat with the following definition.

Definition 2.7. *We define*

$$\mathsf{Bnft}[R_a, B_b] = \left(\sum_{i \in [a,b]} c_i - \sum_{i \in [a,b)} c'_i \right) - c(R_a, B_b),$$

$$\mathsf{Bnft}[B_a, R_b] = \left(\sum_{i \in [a,b)} c'_i - \sum_{i \in (a,b)} c_i \right) - c(B_a, R_b),$$

and, for x and y the same color, $\mathsf{Bnft}[x, y] = -\infty$.

It is immediate that $\mathsf{Bnft}[x, y] > 0$ if and only if $x \to y$ is a candidate; in fact, $\mathsf{Bnft}[x, y]$ measures the benefit (i.e., the reduction in cost) of using $x \leftrightarrow y$ as a minimal jumper instead of the greedy matching on $[x, y]$.

The next lemma forms the basis for the correctness of the algorithm given in section 3 for the serial transitive closure problem. The general idea is that the algorithm will scan the nodes in tour order until at least one candidate is found and then, according to Lemma 2.8, the algorithm will choose an interval (x_ℓ, x_k) to greedily match. Once the interval (ℓ, k) has been greedily matched, the algorithm need only solve the induced matching problem on the remaining nodes.

Lemma 2.8. *Let G be a balanced quasi-convex tour matching problem. Let $1 < k \leq N$ and suppose $\mathsf{Bnft}[x_i, x_j] \leq 0$ for all $1 \leq i < j < k$. Further suppose that $m \overset{\text{def}}{=} \max\{\mathsf{Bnft}[x_i, x_k] : i < k\} > 0$ and let $\ell \overset{\text{def}}{=} \max\{i < k : \mathsf{Bnft}[x_i, x_k] = m\}$. Then every minimum-cost, crossing-free matching is greedy on (x_ℓ, x_k).*

Proof. The proof is, in essence, an iteration of Lemma 2.6. We argue by induction on k. Let G, k, m, and ℓ satisfy the hypothesis of the lemma. Let $s = \max\{i < k : \mathsf{Bnft}[x_i, x_k] > 0\}$, so $x_s \to x_k$ is a minimal candidate. By Lemma 2.6, any minimum-cost, crossing-free solution for G is greedy on the interval (x_s, x_k).

Hence, it will suffice to let G' be the matching problem obtained from G by discarding the nodes $x_{s+1}, \ldots, x_{k-1}$ and prove that any minimum-cost, crossing-free solution for G' is greedy on $(x_\ell, x_s]$. If $\ell = s$, there is nothing to prove, so we assume $\ell < s$. Note that x_k is now the $(s+1)$st node in the G' tour order. We use Bnft' to denote the Bnft function for G'.

We claim the following:

(i) If $1 \le i < j \le s$, $\mathsf{Bnft}'[x_i, x_j] = \mathsf{Bnft}[x_i, x_j]$.

(ii) If $1 \le i \le s$, $\mathsf{Bnft}'[x_i, x_k] = \mathsf{Bnft}[x_i, x_k] - \mathsf{Bnft}[x_s, x_k]$.

Claim (i) is immediate from the definition of Bnft. The intuitive meaning of (ii) is that the benefit of using the jumper $x_i \leftrightarrow x_k$ is reduced by the benefit already obtained from the jumper $x_s \leftrightarrow x_k$. We formally prove (ii) for the case that x_i and x_s are red and x_k is blue; the opposite colored case has a similar proof. Assume $x_i = R_a$, $x_s = R_b$, and $x_k = B_c$. Then

$$\mathsf{Bnft}'[R_a, B_c] = \sum_{\ell \in [a,b)} c_i + c(R_b, B_c) - \sum_{\ell \in [a,b)} c_i' - c(R_a, B_c),$$

$$\mathsf{Bnft}[R_a, B_c] = \sum_{\ell \in [a,c]} c_i - \sum_{\ell \in [a,c)} c_i' - c(R_a, B_c),$$

$$\mathsf{Bnft}[R_b, B_c] = \sum_{\ell \in [b,c]} c_i - \sum_{\ell \in [b,c)} c_i' - c(R_b, B_c).$$

From these three equations claim (ii) follows immediately.

Now let $m' = \max\{\mathsf{Bnft}'[x_i, x_k] : i < s\}$. By claim (ii), $m' = m - \mathsf{Bnft}[x_s, x_k]$; since $\ell < s$, $m' > 0$. Likewise, $\ell = \max\{i < s : \mathsf{Bnft}'[x_i, x_k] = m'\}$. Thus, by the induction hypothesis, any minimum-cost solution for G' is greedy on $(x_\ell, x_s]$ and Lemma 2.8 is proved. ∎

Definition 2.9. *The Δ function is defined by*

$$\Delta[R_a, R_b] = \sum_{\ell \in [a,b)} c_\ell - \sum_{\ell \in [a,b)} c_\ell',$$

$$\Delta[B_a, B_b] = \sum_{\ell \in [a,b)} c_\ell' - \sum_{\ell \in (a,b]} c_\ell.$$

Lemma 2.10.

(i) $\mathsf{Bnft}[R_a, B_c] > \mathsf{Bnft}[R_b, B_c]$ *if and only if* $c(R_a, B_c) - c(R_b, B_c) < \Delta[R_a, R_b]$.

(ii) $\mathsf{Bnft}[B_a, R_c] > \mathsf{Bnft}[B_b, R_c]$ *if and only if* $c(B_a, R_c) - c(B_b, R_c) < \Delta[B_a, B_b]$.

Lemma 2.10 follows immediately from the definitions.

Lemma 2.11. *Let u, v, x, y be in tour order with nodes u and v of one color and x and y of the other color. Then*

$$\mathsf{Bnft}[u, x] > \mathsf{Bnft}[v, x] \quad \Rightarrow \quad \mathsf{Bnft}[u, y] > \mathsf{Bnft}[v, y].$$

Proof. By Lemma 2.10, $\mathsf{Bnft}[u, x] > \mathsf{Bnft}[v, x]$ is equivalent to $c(u, x) - c(v, x) < \Delta[u, v]$, and $\mathsf{Bnft}[u, y] > \mathsf{Bnft}[v, y]$ is equivalent to $c(u, y) - c(v, y) < \Delta[u, v]$. Now, by quasi convexity, $c(u, x) - c(v, x) \ge c(u, y) - c(v, y)$, which suffices to prove the lemma. ∎

Let R_a and R_b be distinct red nodes. The previous two lemmas show that if there is any node B_c (with R_a, R_b, and B_c in tour order) such that $\mathsf{Bnft}[R_a, B_c]$ is greater than $\mathsf{Bnft}[R_b, B_c]$, then the first such B_c is the $\Delta[R_a, R_b]$-crossover point of

R_a and R_b. We shall denote this first B_c, if it exists, by $\chi[R_a, R_b]$; if it does not exist, then $\chi[R_a, R_b]$ is said to be undefined. Similarly, $\chi[B_a, B_b]$ is defined to the be the $\Delta[B_a, B_b]$-crossover point of B_a and B_b, and, if defined, is the first R_c where $\mathsf{Bnft}[B_a, R_c]$ is greater than $\mathsf{Bnft}[B_b, R_c]$.

We now assume that we have a procedure $\Omega(x, y, z)$, which, given nodes x, y, z in tour order, returns "True" if $\chi[x, y] \prec_y \chi[y, z]$ and returns "False" if $\chi[y, z] \prec_y \chi[x, y]$. (If neither condition holds, then $\Omega(x, y, z)$ may return an arbitrary truth value.) If the weak analyticity condition holds, then Ω is constant-time computable. Without this assumption, Ω is $O(\log N)$ time computable since Lemma 2.11 allows $\chi[-, -]$ to be computable by binary search.

The general idea of the algorithm given in section 3 below is that it will scan the nodes in tour order searching for candidates. Whenever a node is reached that is the head of a candidate, the algorithm will take the candidate specified in Lemma 2.8 (the one that was denoted $x_\ell \to x_k$) and greedily match the nodes in its interior. The greedily matched nodes are then dropped from consideration and the algorithm resumes its search for a candidate. Suppose the u and v are two nodes already scanned in this process that are being remembered as potential endpoints of candidates. Lemma 2.10 tells us that if a node x is found where $\mathsf{Bnft}[u, x] > \mathsf{Bnft}[v, x]$, then at all succeeding nodes y, $\mathsf{Bnft}[u, y] > \mathsf{Bnft}[v, y]$. By the criterion of Lemma 2.8, this means that after the node x is found, there is no further reason to consider candidates that begin at node v, since any candidate $v \to y$ would be subsumed by the better candidate $u \to y$.

To conclude this section we describe the algorithm in very general terms; in section 3 we give the precise specification of the algorithm. The algorithm scans nodes (starting with node x_1, say) and maintains three lists. The first list, $\mathcal{M}$, contains the nodes in tour order which have been examined so far. The second list, $\mathcal{L}^{-1}$, contains all the red nodes that need to be considered as potential endpoints of candidates (so $\mathcal{L}^{-1}$ is guaranteed to contain all the nodes satisfying the criterion of Lemma 2.8). The third list, $\mathcal{L}^1$, similarly contains all the blue nodes that need to be considered as potential endpoints of candidates. At any point during the scan, the lists will be of the form

$$\mathcal{M} = x_1, \ldots, x_{r-1},$$

$$\mathcal{L}^{-1} = R_{a_1}, \ldots, R_{a_p},$$

$$\mathcal{L}^1 = B_{b_1}, \ldots, B_{b_q},$$

with $\mathcal{L}^{-1}$ and $\mathcal{L}^1$ subsequences of $\mathcal{M}$. The following five conditions will be maintained during execution:

(i) $x_1, \ldots, x_{r-1}$ are the nodes scanned but not matched and are in tour order, and there are no candidates $x_i \to x_j$ with $1 \leq i < j < r$.

(ii) x_{r-1} precedes $\chi[R_{a_{p-1}}, R_{a_p}]$ in tour order.

(iii) For all $1 \leq i \leq p-2$, $\Omega(R_{a_i}, R_{a_{i+1}}, R_{a_{i+2}})$ is false.

(iv) For all $1 \leq i \leq q-2$, $\Omega(B_{b_i}, B_{b_{i+1}}, B_{b_{i+2}})$ is false.

(v) At any possible future node x_k following x_{r-1} such that x_k is the first point where a candidate is discovered; if the x_ℓ which satisfies Lemma 2.8 is among $x_1, \ldots, x_{r-1}$ then it is already on the list $\mathcal{L}^{-1}$ or $\mathcal{L}^1$ (depending on which color it is).

When scanning the next node x_r, the algorithm must do the following (we assume x_r is blue; similar actions are taken for red nodes):

(β) While $p \geq 2$ and $\mathsf{Bnft}[R_{a_{p-1}}, x_r] > \mathsf{Bnft}[R_{a_p}, x_r]$, pop R_{a_p} from $\mathcal{L}^{-1}$ and decrement p.

(γ) If $\mathsf{Bnft}[R_{a_p}, x_r] > 0$, greedily match nodes in the interval (R_{a_p}, x_r). The matched nodes are discarded from the lists $\mathcal{M}$, $\mathcal{L}^{-1}$, and $\mathcal{L}^1$ (the remaining nodes are to be implicitly renumbered at this point).

(δ) While $q \geq 2$ and $\Omega(B_{a_{q-1}}, B_{a_q}, x_r)$, pop B_{a_q} from $\mathcal{L}^1$ and decrement q. Then push x_r onto the end of $\mathcal{L}^1$ (and increment q).

Step (β) is justified by recalling that if x_r is past $\chi[R_{a_{p-1}}, R_{a_p}]$, then R_{a_p} may be removed from consideration as an endpoint of a candidate (by Lemma 2.8).

Step (δ) is justified as follows: suppose $R_i = \chi[B_{a_{q-1}}, B_{a_q}]$ equals or precedes $R_j = \chi[B_{a_q}, x_r]$ (using tour order, beginning at B_{a_q}). Then at any future candidate endpoint x_k, either x_k follows or equals R_i, in which case $\mathsf{Bnft}[B_{a_{q-1}}, x_k]$ is greater than $\mathsf{Bnft}[B_{a_q}, x_k]$, or x_k precedes R_j, in which case, $\mathsf{Bnft}[x_r, x_k]$ is greater than $\mathsf{Bnft}[B_{a_q}, x_k]$. Thus B_{a_q} will never be the starting endpoint of a candidate satisfying the criteria of Lemma 2.8, and we may drop it from consideration.

To justify step (γ) we must show that the candidate $R_{a_p} \to x_r$ satisfies the criteria from Lemma 2.8; in view of the correctness of the rest of the algorithm, for this it will suffice to show that $\mathsf{Bnft}[R_{a_i}, x_r] \leq \mathsf{Bnft}[R_{a_p}, x_r]$ for all $1 \leq i < p$. For this, note that step (β) and condition (iii) above ensure that x_r precedes $\chi[R_{a_i}, R_{a_{i+1}}]$ for all $1 \leq i < p$. This, in turn, implies $\mathsf{Bnft}[R_{a_i}, x_r] \leq \mathsf{Bnft}[R_{a_{i+1}}, x_r]$ for all i, which proves the desired inequality.

After the algorithm has scanned all the nodes once, it will have found and processed all candidates $x_i \to x_j$ where $i < j$. However, since the tour is circular, it is necessary to process candidates $x_i \to x_j$ with $i > j$. At the end of the first scan, the list $\mathcal{M}$ consists of all nodes $x_1, \ldots, x_n$ which have not been matched yet and $\mathcal{L}^{-1}$ and $\mathcal{L}^1$ contain nodes $R_{a_1}, \ldots, R_{a_p}$ and $B_{b_1}, \ldots, B_{b_q}$, as usual. During the second scan, the algorithm is searching for any candidates of the form $R_{a_i} \to B_j$ with $j < a_i$ or of the form $B_{a_i} \to R_j$ with $j \leq a_i$ (and only for such candidates). To process a node during the second scan, the algorithm pops x_1 off the left end of $\mathcal{M}$, implicitly renames x_1 to x_n and the rest of the nodes x_i to x_{i-1}, sets $r = n$, and does step (α) (still assuming x_r is blue):

(α) If x_r equals B_{b_1}, then pop B_{b_1} from the list $\mathcal{L}^1$ and implicitly renumber $\mathcal{L}^1$, decrementing q.

It then does steps (β)–(δ), except that in step (δ), the node x_r is not added to the end of $\mathcal{L}^1$. The reason for step (α) is that once a node B_{b_i} is encountered on the second scan, B_{b_i} is no longer a possible starting endpoint for a candidate. The reason for not adding x_r to the end of $\mathcal{L}^1$ in step (δ) is that it cannot be the starting endpoint of a candidate, because any such candidate would have already been found earlier.

The second scan will stop as soon as both $\mathcal{L}$ lists become empty. At this point no candidates remain and a greedy matching may be used for the remaining nodes in the $\mathcal{M}$ list.

The actual description of the algorithm with an efficient implementation is given in section 3, and it is there proved that the algorithm is linear time with the weak analyticity condition and $O(N \log N)$ time otherwise. Although we described steps (α)–(δ) only for blue x_r above, the algorithm in section 3 uses a toggle ψ to handle both colors with the same code. Finally, one more important feature of the algorithm is the way in which it computes the values of the Bnft function and of the $\Delta[x, y]$ function; it uses intermediate values $I[x]$ which are defined as follows.

DEFINITION 2.12. *The $I[x]$ function is defined by*

$$I[R_a] = \Delta[R_1, R_a],$$

$$I[B_a] = I[R_a] + c(R_a, B_a).$$

Note that $I[R_{a+1}] = I[B_a] - c(B_a, R_{a+1})$.

It is immediate from the definitions that, if x, y are tour order (starting from x_1), then

$$\begin{aligned}
\Delta[x,y] &= I[y] - I[x] && \text{for } x \text{ and } y \text{ red,}\\
\Delta[x,y] &= I[x] - I[y] && \text{for } x \text{ and } y \text{ blue,}\\
\mathsf{Bnft}[x,y] &= I[y] - I[x] - c(x,y) && \text{for } x \text{ red, } y \text{ blue,}\\
\mathsf{Bnft}[x,y] &= I[x] - I[y] - c(x,y) && \text{for } x \text{ blue, } y \text{ red.}
\end{aligned}$$

These equalities permit the values of Δ and Bnft to be computed in constant time from the values of $I[-]$. Also, it is important to note that only the relative $I[-]$ values are needed; in other words, it is OK if the $I[-]$ values are shifted by a constant additive constant, since we always use the difference between two $I[-]$ values.

The $I[-]$ function is not only easy to compute but also provides an intuitive graphical means of understanding the above lemmas and algorithm description. For example, in Figure 2.1, $R_1 \to B_3$ is a (minimal) candidate whereas $R_1 \to B_1$ and $R_1 \to B_2$ are not candidates. In Figure 2.2(a), the node B_3 is the relative crossover, $\chi[R_1, R_2]$, of R_1 and R_2; on the other hand, in Figure 2.2(b), the relative crossover does not exist. Figure 2.3(a) shows an example where $\Omega(R_1, R_2, R_3)$ is true and Figure 2.3(b) shows an example where $\Omega(R_1, R_2, R_3)$ is false.

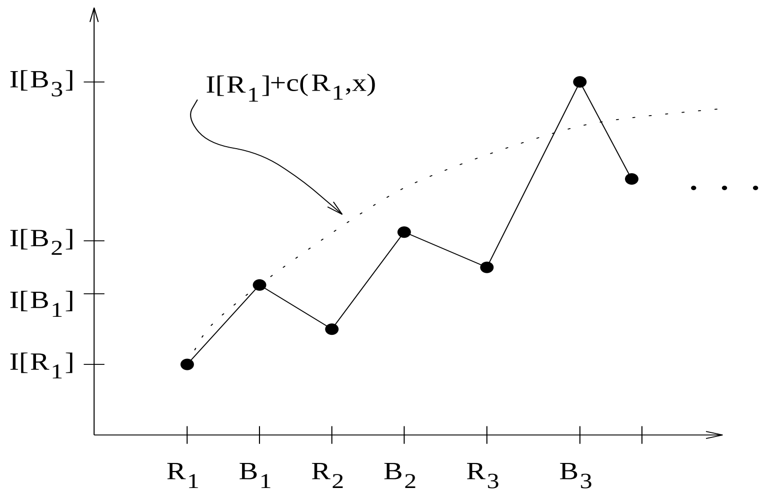

FIG. 2.1. *$R_1 \to B_3$ is a candidate as $I[R_1] + c(R_1, B_3) < I[B_3]$, which is equivalent to* $\mathsf{Bnft}[R_1, B_3] > 0$.

2.3. Proof of Lemma 2.6. By symmetry it will suffice to prove part (i). Since the lemma is trivial in case $a = b$, we assume $a \neq b$. Let σ be a crossing-free minimum-cost matching; we must prove that σ is greedy on (R_a, B_b). By the crossing freeness of σ and by the fact that $R_a \to B_b$ is a *minimal* candidate, σ does not contain any jumper with both endpoints in $[R_a, R_b]$, except possibly $R_a \leftrightarrow B_b$ itself. If $R_a \leftrightarrow B_b$ is in σ, then the same reasoning shows that σ is greedy on (R_a, B_b); so we suppose that $R_a \leftrightarrow B_b$ is not in σ. Since we are dealing (w.l.o.g.) with balanced tours, we may assume that $b = N$, by renumbering nodes if necessary.

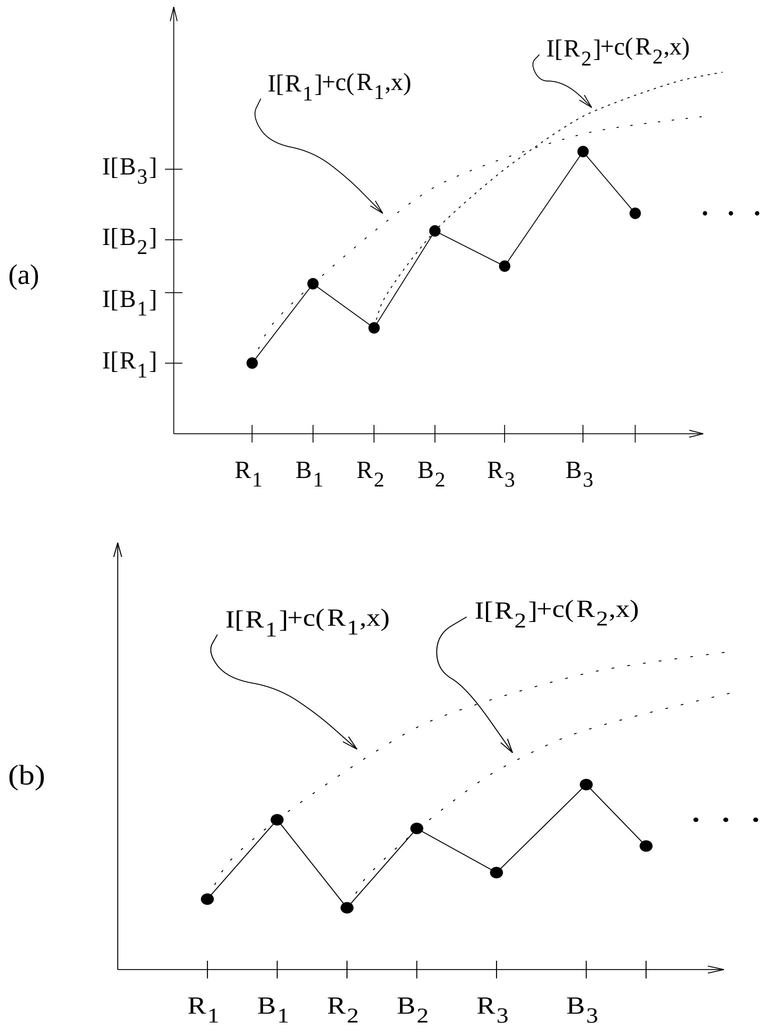

FIG. 2.2. *Illustrations of the relative crossover,* $\chi[R_1.R_2]$, *of* R_1 *and* R_2. *In* (a), B_3 *is* $\chi[R_1, R_2]$, *since it is the first node* x *to satisfy* $I[R_2]+c(R_2,x) > I[R_1]+c(R_1,x)$. *In* (b), *the relative crossover does not exist.*

Claim (i). $R_a \leftrightarrow B_a$ is not in σ.

Suppose, for a contradiction, that $R_a \leftrightarrow B_a$ is in σ. Let v be the least value such that $R_v \leftrightarrow B_q$ is in σ for some $q < a < v$. Note that such a v, $a < v \le N$, must exist since there are no jumpers in $[R_a, B_N]$ and since σ is not greedy on $[R_a, B_N]$ (it cannot be greedy on $[R_a, B_N]$, since $R_a \to B_N$ is a candidate). By choice of v, σ is greedy on $[R_a, R_v)$. These edges in the matching σ are represented by edges drawn above the line in Figure 2.4(a). Since $R_a \to B_N$ is a minimal candidate, $\mathsf{Bnft}[R_a, B_N] > \mathsf{Bnft}[R_v, B_N]$, so Lemma 2.10 implies

$$\sum_{i\in[a,v)} c_i - \sum_{i\in[a,v)} c'_i > c(R_a, B_N) - c(R_v, B_N).$$

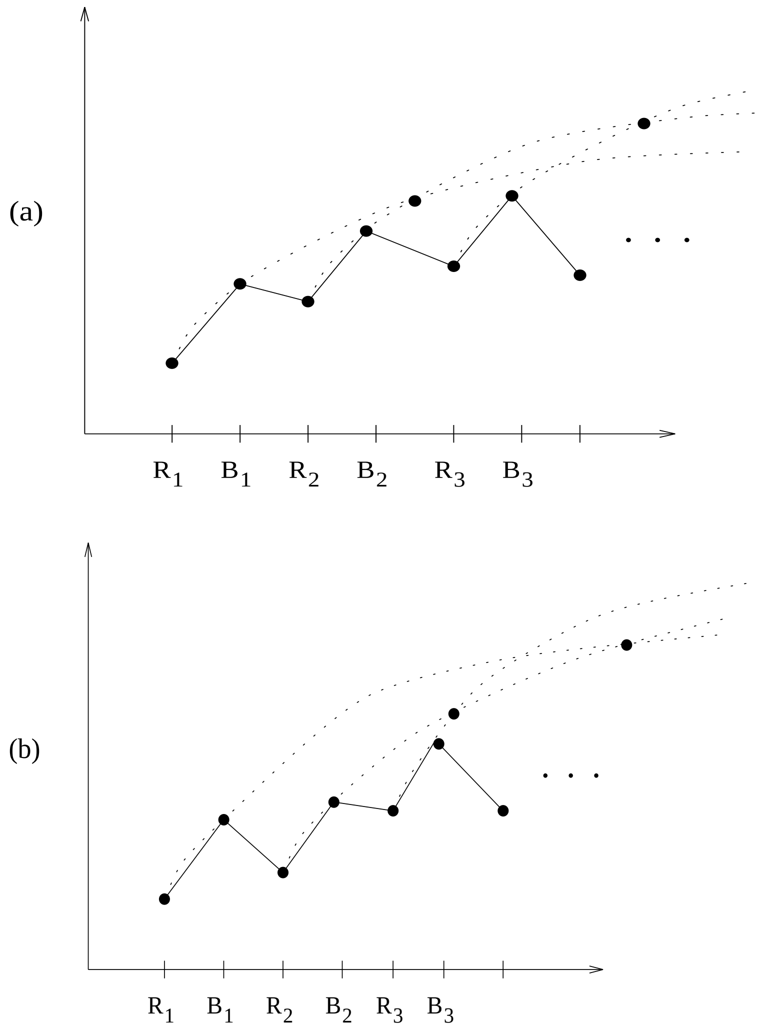
(a)
R_1 B_1 R_2 B_2 R_3 B_3
(b)
R_1 B_1 R_2 B_2 R_3 B_3

FIG. 2.3. $\Omega(R_1, R_2, R_3)$ *is true in* (a) *and false in* (b).

Since R_a, R_v, B_N, and B_q are in tour order, quasi convexity implies

$$c(R_a, B_N) - c(R_v, B_N) \geq c(R_a, B_q) - c(R_v, B_q).$$

Combining these inequalities yields

$$\sum_{i \in [a,v)} c_i + c(R_v, B_q) > \sum_{i \in [a,v)} c'_i + c(R_a, B_q). \tag{2.1}$$

Let σ' be the matching obtained from σ by replacing the jumper $B_q \leftrightarrow R_v$ and the greedy matching on $[R_a, R_v)$ with the edge $B_q \leftrightarrow R_a$ and the greedy matching on $(R_a, R_v]$. The new edges in σ' are drawn below the line in Figure 2.4(a). By (2.1), σ' has cost strictly less than the cost of σ, which is a contradiction.

Claim (ii). $R_N \leftrightarrow B_N$ is not in σ.

Claim (ii) is proved by an argument similar to Claim (i). Alternatively, reverse the colors and the tour order and Claim (ii) is a version of Claim (i).

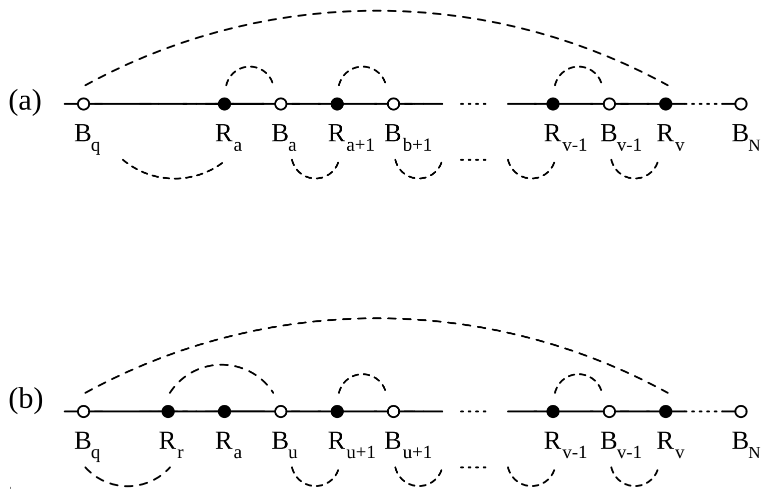

FIG. 2.4. *Illustrations of Claims* (i) *and* (iii) *from the proof of Lemma* 2.6*. The edges above the lines represent edges in the presumed minimum-cost matching σ; these are replaced by the edges below the line in the lower-cost matching σ'.*

Claim (iii). The matching σ is greedy on (R_a, B_b).

Suppose for a contradiction that σ is not greedy on (R_a, B_b). In view of Claims (i) and (ii) and since σ has no jumpers in $[R_a, B_N]$, this means that there exist u and v such that u is the least value, such that σ contains $B_u \leftrightarrow R_r$ with $r < a \le u$, and v is the least value, such that σ contains $R_v \leftrightarrow B_q$ with $q < a \le v$. Namely, let u be the least value $\ge a$ such that $B_u \leftrightarrow R_{u+1}$ is not in σ and v be the least value $> u$ such that $R_v \leftrightarrow B_v$ is not in σ. For these choices of u and v, it must be that $q < r < a \le u \le v \le N$ and that σ is greedy on $[B_a, R_u]$ and $[R_{u+1}, B_{v-1}]$. These edges in the matching σ are represented by edges drawn above the line in Figure 2.4(b).

Since $R_a \to B_N$ is a candidate,

$$\sum_{i\in[a,N]} c_i > c(R_a, B_N) + \sum_{i\in[a,N)} c'_i.$$

And since it is minimal, neither $R_a \to B_u$ nor $R_v \to B_N$ are candidates; i.e.,

$$\sum_{i\in[a,u]} c_i \le c(R_a, B_u) + \sum_{i\in[a,u)} c'_i,$$

$$\sum_{i\in[v,N]} c_i \le c(R_v, B_N) + \sum_{i\in[v,N)} c'_i.$$

Combining these three inequalities gives

$$\sum_{i\in(u,v)} c_i + c(R_a, B_u) + c(R_v, B_N) > \sum_{i\in[u,v)} c'_i + c(R_a, B_N). \tag{2.2}$$

Since R_a, R_v, B_N, B_q and R_r, R_a, B_u, B_q are in tour order, quasi convexity implies the two inequalities

$$c(R_a, B_q) + c(R_v, B_N) \le c(R_a, B_N) + c(R_v, B_q),$$

$$c(R_r, B_q) + c(R_a, B_u) \leq c(R_r, B_u) + c(R_a, B_q)$$

which combine to yield

$$\begin{aligned} c(R_a, B_N) - c(R_a, B_u) - c(R_v, B_N) \\ \geq c(R_r, B_q) - c(R_r, B_u) - c(R_v, B_q). \end{aligned} \tag{2.3}$$

Using (2.2) and (2.3) gives the inequality

$$\sum_{i \in (u,v)} c_i + c(R_r, B_u) + c(R_v, B_q) > \sum_{i \in [u,v)} c'_i + c(R_r, B_q).$$

Let σ' be the matching obtained from σ by replacing the jumpers $R_r \leftrightarrow B_u$, $R_v \leftrightarrow B_q$, and the greedy matching on (B_u, R_v) with the edge $R_r \leftrightarrow B_q$ and the greedy matching on $[B_u, R_v]$. The new edges in σ' are drawn below the line in Figure 2.4(b). The last inequality above says that σ' has cost strictly less than the cost of σ, which is a contradiction. $\square$

3. The algorithm. In this section, we give the actual algorithm for the main theorems. The correctness of the algorithm follows from the development in section 2.2. With Kanzelberger and Robinson, we have developed efficient implementations in ANSI-C of all the algorithms described below [5].[4]

3.1. Preliminaries. As mentioned above, the algorithm maintains three lists of nodes called deques (for "double ended queues," since we will have to access both ends of the lists). The three deques are the "main" deque $\mathcal{M}$ and two "left" deques $\mathcal{L}^1$ and $\mathcal{L}^{-1}$. The latter two are called "left deques" since they contain possible left endpoints for candidates. The deques will be updated by *push-right* operations which add a new node to the right end, by *pop-right* operations which pop the rightmost node off the deque, and by *pop-left* operations. However, *push-left* operations are never required. Deque operations can be efficiently implemented by using contiguous memory locations to store the deque elements and maintaining pointers to the left and right endpoints; each deque operation can then be performed in constant time. For our algorithm, it will suffice to reserve enough space for $2N$ deque elements (with no possibility that a deque will grow leftward since push-left's are not used).

Subscripts R, L, and $R{-}1$ are used to select the rightmost item, leftmost item, and the item preceding the rightmost, respectively. So $\mathcal{L}^{-1}_L$ refers to the leftmost element of $\mathcal{L}^{-1}$, $\mathcal{M}_{R-1}$ refers to the item just before the rightmost member of $\mathcal{M}$, etc. Each deque element is actually a pair, for example, $\mathcal{M}_R = (X, I)$; the first entry X of the pair is a node and the second entry I is a numerical value, namely $I = I[X]$ as defined in section 2.2. To simplify notation, we shall use the same notation for a deque element as for the node which is its first component. Thus, $\mathcal{M}_L$ also denotes the node which is its first component. We write $I[\mathcal{M}_L]$ to denote its second (numerical) component. Similar conventions apply to the $\mathcal{L}^{\pm 1}$ deques. To simplify our presentation of the algorithm, we deal with boundary effects by augmenting the definition of primitive operations as necessary. For example, accessing a nonexistent deque element will return an *undefined* indicator $\emptyset$ and, in general, functions of undefined operands are false or zero (in particular, the cost function $c(-,-)$ and the $I[-]$ functions return zero if they have $\emptyset$ as an argument).

[4]These C implementations are also available electronically from the authors or can currently be obtained by anonymous ftp from `math.ucsd.edu` or `ftp.nj.nec.com`

Function Input() returns the next vertex from an imagined input tape, which moves in the forward direction only and is assumed to hold a balanced alternating color tour. When the tape's end is reached, "*undefined*" is returned. Procedure Output() is used to write an individual matching to an imagined output tape. They are written as discovered but can easily be output in tour order (with only an extra $O(N)$ time computation).

To use the same code for red nodes and blue nodes, a variable ψ tracks vertex color by toggling between -1 and 1. Our convention is that $\psi = 1$ corresponds to blue and $\psi = -1$ to red.

3.2. Narrative description of the algorithm. Initialization consists of setting the three deques to be empty and setting the color toggle $\psi := -1$.

The algorithm first reads nodes from the input and pushes them onto the right end of the $\mathcal{M}$ deque, and then twice scans the nodes in tour order. During the two scans, nodes are popped from the left end of $\mathcal{M}$ and then pushed onto its right end.[5] In addition, while processing a node some nodes may be popped off the right end of $\mathcal{M}$ to be matched. It will always be the case that $\mathcal{M}$ contains a sequence of contiguous nodes in tour order and that the node currently being scanned immediately follows the (formerly) rightmost element of $\mathcal{M}$.

The variable ψ will be maintained as a color toggle, so that ψ is equal to -1 if the node currently being processed is red and to 1 if the current node is blue. The algorithm used for pushing an element onto the right end of $\mathcal{M}$ follows.

Algorithm 3.1. *This procedure pushes a vertex X onto the right of the $\mathcal{M}$ deque and computes the corresponding $I[X]$ value which is pushed along with X.*

```
**procedure** *Push_Main* $(X)$
    $I := I[\mathcal{M}_R] + \psi \cdot c(\mathcal{M}_R, X)$
    *push-right* $(X, I)$ *onto* $\mathcal{M}$
    *return*()
```

Algorithm 3.1 merely computes the $I[-]$ value for a node X and pushes the node and its $I[-]$ value on the right end of M. To justify the computation of the value of $I[X]$, note that if X is blue, then $\psi = 1$ and $I[X]$ was defined to equal $I[\mathcal{M}_R] - c(\mathcal{M}_R, X)$; whereas, if X is red then $\phi = -1$ and $I[X]$ equals $I[\mathcal{M}_R] + c(\mathcal{M}_R, X)$. (Unless $\mathcal{M}$ is empty, in which case, $I[X] = 0$.)

Once the current node has been pushed onto the right end of $\mathcal{M}$, the following code implements step (β) from section 2.2:

```
**while** $c(\mathcal{L}^{-\psi}_{R-1}, \mathcal{M}_R) - c(\mathcal{L}^{-\psi}_{R}, \mathcal{M}_R) < \psi \cdot (I[\mathcal{L}^{-\psi}_{R}] - I[\mathcal{L}^{-\psi}_{R-1}])$
    *pop-right* $\mathcal{L}^{-\psi}$
```

To justify the correctness of the **while** condition, suppose that the currently scanned node is red, so $\phi = -1$. By Lemma 2.10, $\mathsf{Bnft}[\mathcal{L}^{-\psi}_{R-1}, \mathcal{M}_R] > \mathsf{Bnft}[\mathcal{L}^{-\psi}_{R}, \mathcal{M}_R]$ if and only if $c(\mathcal{L}^{-\psi}_{R-1}, \mathcal{M}_R) - c(\mathcal{L}^{-\psi}_{R}, \mathcal{M}_R) < \Delta[\mathcal{L}^{-\psi}_{R-1}, \mathcal{L}^{-\psi}_{R}]$. Furthermore, $\Delta[\mathcal{L}^{-\psi}_{R-1}, \mathcal{L}^{-\psi}_{R}]$ is equal to $\psi \cdot (I[\mathcal{L}^{-\psi}_{R}] - I[\mathcal{L}^{-\psi}_{R-1}])$ since $\mathcal{L}^{-\psi}$ contains blue nodes and $\psi = -1$ (by the equalities at the end of section 2.2). In this case, $\mathcal{M}_R$ is past the crossover point of $\mathcal{L}^{-\psi}_{R-1}$ and $\mathcal{L}^{-\psi}_{R}$, so $\mathcal{L}^{-\psi}_{R}$ may be discarded from consideration as a left endpoint of a candidate. A similar calculation justifies the case when the current node is blue.

[5]For line-like tours, only the first scan is needed; however, we treat only the more general (circular) case.

To implement step (γ), the following code is used:

```
**if** $c(\mathcal{M}_R, \bar{\mathcal{L}}^{\psi}_R) < \psi \cdot (I[\mathcal{M}_R] - I[\bar{\mathcal{L}}^{\psi}_R])$
    X := *pop-right* $\mathcal{M}$
    **while** $\mathcal{M}_R \neq \bar{\mathcal{L}}^{\psi}_R$
        *Match_Pair*()
    *Push_Main*(X)
```

where Match_Pair is defined below. The above **if** statement checks whether $\bar{\mathcal{L}}^{\psi}_R \to \mathcal{M}_R$ is a candidate; if so, the algorithm greedily assigns edges to nodes in the interior of the candidate (where "greedily" means with respect to the nodes that have not already been assigned). Before the greedy assignment is started, the rightmost entry is popped from $\mathcal{M}$ and is saved as X to pushed back on the right end afterwards. There are two reasons for this: first, this gets the current node X out of the way of Match_Pair's operation, and second, and more importantly, when X is pushed back onto $\mathcal{M}$, the $I[-]$ value for the current node is recomputed so as to be correct for the reduced matching problem in which the greedily matched nodes are no longer present. Match_Pair is the following procedure:

```
**procedure** *Match_Pair*()
    *Output* ("$\mathcal{M}_{R-1} \leftrightarrow \mathcal{M}_R$")
    *pop-right* $\mathcal{M}$
    **if** $\mathcal{M}_R = \mathcal{L}^{\psi}_R$
        *pop-right* $\mathcal{L}^{\psi}$
    *pop-right* $\mathcal{M}$
    *return*()
```

The procedure Match_Pair assigns a jumper $\mathcal{M}_{R-1} \leftrightarrow \mathcal{M}_R$ and discards a matched node from the deque $\mathcal{L}^{\psi}$ if it appears there. Because of the **while** condition controlling calls to Match_Pair, it is not possible for a matched node to occur in $\bar{\mathcal{L}}^{\psi}$, so we do not check for this condition.

To implement step (δ), the following code is used:

```
**while** $\Omega(\mathcal{L}^{\psi}_{R-1}, \mathcal{L}^{\psi}_R, \mathcal{M}_R)$ = "Yes"
    pop-right $\mathcal{L}^{\psi}$
push-right $\mathcal{M}_R$ onto $\mathcal{L}^{\psi}$          (without popping $\mathcal{M}_R$)
```

That completes the description of how nodes are processed during the first scan. As mentioned earlier, the last instruction (the *push-right*) is omitted from step (δ) during the second scan. Other than this, the processing for steps (β)–(δ) is identical in the two scans.

One potentially confusing aspect of the second scan is that the $I[-]$ values are no longer actually the correct $I[-]$ values; for example, it is no longer the case that $I[\mathcal{M}_L]$ is necessarily equal to zero. Strictly speaking, the $I[-]$ values all shift by an additive constant when an entry is popped from the left end of $\mathcal{M}$; however, it is not necessary to implement this shift, since the algorithm only uses differences between $I[-]$ values. The end result is that nothing special needs to be done to the I values when we pop-left $\mathcal{M}$.

After both scans are completed, any remaining nodes may be greedily matched. As discussed above, there are two possible greedy matchings and both have the same (optimal) cost. Thus either one may be used; the algorithm below just calls

Match_Pair repeatedly to assign one of these greedy matchings.

ALGORITHM 3.2. *This is the matching algorithm for balanced quasi-convex tours. All variables are global.*

```
"Initialization"
$\mathcal{M}, \mathcal{L}^{-1}, \mathcal{L}^{1} := \emptyset$
$\psi := -1$
"Read Input into the $\mathcal{M}$ deque"
while $[X := Input()] \neq \emptyset$
    $Push\_Main\ (X)$
    $\psi := -\psi$
"The First Scan"
while $\mathcal{L}^{\psi}$ is empty or $\mathcal{M}_L \neq \mathcal{L}^{\psi}_L$
    $X :=$ pop-left $\mathcal{M}$
    $Process\_Node()$
    push-right $\mathcal{M}_R$ onto $\mathcal{L}^{\psi}$
    $\psi := -\psi$
"The Second Scan"
while $\mathcal{L}^{-1}$ and $\mathcal{L}^{1}$ are not both empty
    $X :=$ pop-left $\mathcal{M}$
    if $X = \mathcal{L}^{\psi}_L$
        pop-left $\mathcal{L}^{\psi}$
    $Process\_Node()$
    $\psi := -\psi$
"Windup Processing"
while $\mathcal{M}$ is not empty
    $Match\_Pair()$
Exit.
procedure $Process\_Node()$
    $Push\_Main(X)$
    while $c(\mathcal{L}^{-\psi}_{R-1}, \mathcal{M}_R) - c(\mathcal{L}^{-\psi}_{R}, \mathcal{M}_R) < (I[\mathcal{L}^{-\psi}_{R}] - I[\mathcal{L}^{-\psi}_{R-1}]) \cdot \psi$
        pop-right $\mathcal{L}^{-\psi}$
    if $c(\mathcal{M}_R, \mathcal{L}^{-\psi}_{R}) < \psi \cdot (I[\mathcal{M}_R] - I[\mathcal{L}^{-\psi}_{R}])$
        $X :=$ pop-right $\mathcal{M}$
        while $\mathcal{M}_R \neq \mathcal{L}^{-\psi}_{R}$
            $Match\_Pair()$
        $Push\_Main(X)$
    while $\Omega(\mathcal{L}^{\psi}_{R-1}, \mathcal{L}^{\psi}_{R}, \mathcal{M}_R)$
        pop-right $\mathcal{L}^{\psi}$
    return
```

The complete matching algorithm is shown as Algorithm 3.2. When interpreting Algorithm 3.2, it is necessary to recall our convention that any predicate of undefined arguments is to be false. This situation can occur in the four **while** and **if** conditions of Process_Node. If $\mathcal{L}^{-\psi}$ is empty, then $\mathcal{L}^{-\psi}_{R}$ is undefined and the two **while** conditions and the first **if** condition are to be false. Similarly, if $\mathcal{L}^{-\psi}$ has < 2 elements, then the first **while** condition is to be false; and if $\mathcal{L}^{\psi}$ has < 2 elements, then the final **while** condition is to be false.

The runtime of Algorithm 3.2 is either $O(N)$ or $O(N \log N)$ depending on whether the weak analyticity condition holds. To see this, note that the initialization and the windup processing both take $O(N)$ time. The loops for each of the two scans are executed $\leq N$ times. Except for the **while** loops, each call to Process_Node takes constant time. The second **while** loop (which calls Match_Pair) is executed more than once only when edges are being output. If the first or third **while** loop is executed more than once, then vertices are being popped from the L stacks. Since $\lfloor \frac{1}{2}N \rfloor$ edges are output and since $O(N)$ vertices are pushed onto the L stacks, each of these **while** loops are executed only $O(N)$ times during the *entire* execution of the algorithm. An iteration of the first or second **while** loop takes constant time, while an iteration of the third **while** loop takes either constant time or $O(\log N)$ time, depending on whether the weak analyticity property holds.

When the weak analyticity condition holds, the Ω predicate typically operates in constant time by computing two theoretical relative crossovers and comparing their positions. This happens, for example, when the tour consists of points lying on a circle, with the cost function equal to Euclidean distance; section 3.3 outlines a constant-time algorithm for this example. Without the weak analyticity condition, the Ω-predicate runs in logarithmic time, by using a binary search of the $\mathcal{M}$ deque. This general (not weakly analytic) case is handled by the generic Ω algorithm discussed in section 3.3.

There are a couple of improvements that can be made to the algorithm which will increase execution speed by a constant factor. First, the calls to Match_Pair made during the "Windup Processing" do not need to check if $\mathcal{M}_R = \mathcal{L}^{\psi}_R$, since $\mathcal{L}^{\psi}$ is empty at this time. Second, if computing the cost function $c(-,-)$ is more costly than simple addition, then it is possible for Push_Main() to use an alternative method during the two scans to compute the cost $c(\mathcal{M}_R, X)$ for nodes X which have just been popped from the left of $\mathcal{M}$ (except for the first one popped from the left in the first scan). Namely, the algorithm can save the old $I[X]$ value for the node X as it is left-popped off the deque $\mathcal{M}$. Then the cost function can be computed by computing the difference between the $I[-]$ value of X and the $I[-]$ of the previous node left-popped from $\mathcal{M}$. This second improvement applies only to the first Push_Main call in Process_Node.

3.3. Algorithms for Ω. During the first scan, the procedure Ω is called (repeatedly) by Process_Node to determine whether $\mathcal{L}^{\psi}_R$ should be popped before the current node $\mathcal{M}_R$ is pushed onto the right end of the $\mathcal{L}^{\psi}$ deque. During the second scan, $\mathcal{M}_R$ is never pushed onto the $\mathcal{L}^{\psi}$ deque; however, using the procedure Ω can allow the $\mathcal{L}^{\psi}$ deque to be more quickly emptied, thus speeding up the algorithm's execution. (However, the use of the Ω could be omitted during the second scan without affecting the correctness of the algorithm.)

When Ω is called, $\mathcal{L}^{\psi}_{R-1}$, $\mathcal{L}^{\psi}_R$, and $\mathcal{M}_R$ are distinct nodes, in tour order, and of the same color. Let $\delta = (I[\mathcal{L}^{\psi}_{R-1}] - I[\mathcal{L}^{\psi}_R]) \cdot \psi$ and $\epsilon = (I[\mathcal{L}^{\psi}_R] - I[\mathcal{M}_R]) \cdot \psi$. Let Y denote the δ-crossover point of $\mathcal{L}^{\psi}_{R-1}$ and $\mathcal{L}^{\psi}_R$, and let Z denote the ϵ-crossover point of $\mathcal{L}^{\psi}_R$ and $\mathcal{M}_R$ (note Y and/or Z may not exist). By definition, Y and Z are both opposite in color from the other three nodes. The procedure Ω must return *True* if Y exists and Z does not, must return *False* if Y does not exist, and, if both exist, must return *True* if $\mathcal{L}^{\psi}_R, Y, Z$ are in tour order, must return *True* if $\mathcal{L}^{\psi}_R, Z, Y$ are in tour order, and may return either value if $Y = Z$.

In this section, we discuss two algorithms for Ω. We first discuss a "generic" algorithm that works for any cost function, regardless of whether the weak analyticity condition holds. This generic Ω procedure is shown as Algorithm 3.3 below. The generic Ω executes a binary search for a node which is at or past one of the crossover points but is not at or past the other. Obviously, if the crossover Y exists, then it exists in the range $(\mathcal{L}^{\psi}_{R}, \mathcal{L}^{\psi}_{R-1})$, and if the crossover Z exists, it is in the range $(\mathcal{M}_R, \mathcal{L}^{\psi}_{R})$. Furthermore, Y cannot exist in the range $(\mathcal{L}^{\psi}_{R}, \mathcal{M}_R)$, since otherwise it would have been popped when Y was reached (by the first **while** loop in an earlier call to Match_Pair). Hence, the binary search may be confined to the range $(\mathcal{M}_R, \mathcal{L}^{\psi}_{R-1})$, provided that *True* is returned in the event that the binary search is unsuccessful.

In Algorithm 3.3, a new notation $\mathcal{M}_k$ is used. This presumes that the $\mathcal{M}$ deque is implemented as an array; the elements of $\mathcal{M}$ fill a contiguous block of the array elements. When we write $\mathcal{M}_k$, we mean the kth entry of the array. The L deques can contain pointers to $\mathcal{M}$ deque entries; in fact, in our preferred implementation, the L deque entries contain only an index for an $\mathcal{M}$ deque entry. Thus the value h can be found in constant time for Algorithm 3.3.

The generic Ω algorithm shown in Algorithm 3.3 takes $O(\log N)$ time since it uses a binary search.

Algorithm 3.3. *This is the generic Ω algorithm which works with any tour, regardless of weak analyticity. Past_Xover_A and Past_Xover_B are boolean-valued variables.*

```
procedure $\Omega(\mathcal{L}^{\psi}_{R-1}, \mathcal{L}^{\psi}_{R}, \mathcal{M}_R)$
    "Nodes $\mathcal{L}^{\psi}_{R-1}$, $\mathcal{L}^{\psi}_{R}$ and $\mathcal{M}_R$ are the same color."

    Let $h$ be the index so that $\mathcal{M}_{h-1}$ is $\mathcal{L}^{\psi}_{R-1}$.
    Let $\ell$ be the index so that $\mathcal{M}_\ell$ is $\mathcal{M}_R$.

    $\delta := (I[\mathcal{L}^{\psi}_{R-1}] - I[\mathcal{L}^{\psi}_{R}]) \cdot \psi$
    $\epsilon := (I[\mathcal{L}^{\psi}_{R}] - I[\mathcal{M}_R]) \cdot \psi$
    "Do binary search of opposite color nodes from $\mathcal{M}_\ell$ to $\mathcal{M}_{h-2}$"
    while $h > \ell + 1$
        $k := \ell + 2\lfloor (h-\ell)/4 \rfloor$
        Past_Xover_A:= $(c(\mathcal{L}^{\psi}_{R-1}, \mathcal{M}_k) - c(\mathcal{L}^{\psi}_{R}, \mathcal{M}_k)) < \delta$
        Past_Xover_B:= $(c(\mathcal{L}^{\psi}_{R}, \mathcal{M}_k) - c(\mathcal{M}_R, \mathcal{M}_k)) < \epsilon$
        if Past_Xover_A
            if Past_Xover_B
                $h := k$
            else
                return(TRUE)
        else
            if Past_Xover_B
                return(FALSE)
            else
                $\ell := k + 2$
    return (TRUE)
```

Next we describe an example of a linear time algorithm for Ω where the weak analyticity condition holds. For this example, we assume that the nodes of the quasiconvex tour lie on the unit circle in the xy plane, the cost function is equal to straight-line Euclidean distance, and the tour proceeds in counterclockwise order around the circle. The Ω algorithm either is given, or computes, the xy coordinates of the three nodes $\mathcal{L}^{\psi}_{R-1}$, $\mathcal{L}^{\psi}_{R}$, and $\mathcal{M}_R$. It then uses a routine Circle_Crossover to find the **theoretical**

δ-crossover Y of $\mathcal{L}^{\psi}_{R-1}$ and $\mathcal{L}^{\psi}_{R}$ and the **theoretical** ϵ-crossover of $\mathcal{L}^{\psi}_{R}$ and $\mathcal{M}_R$. If the theoretical crossover Y exists, it will be in the interval $[\mathcal{L}^{\psi}_{R}, \mathcal{L}^{\psi}_{R-1}]$, and if Z exists, Z will be in the interval $[\mathcal{M}_R, \mathcal{L}^{\psi}_{R}]$. When Y and Z both exist, the Ω procedure returns *True* if $\mathcal{L}^{\psi}_{R}, Y, Z$ are in tour order or any two of these nodes are equal; otherwise the procedure returns *False*.

ALGORITHM 3.4. *This is the algorithm which computes the δ-theoretical crossover point for two nodes lying on the unit circle with cost function equal to Euclidean distance. The inputs are δ and two points (x_1, y_1) and (x_2, y_2) lying on the unit circle. The procedure returns TRUE or FALSE to indicate whether the crossover point exists; if it does exist, it sets (x_3, y_3) equal to the crossover point. There is a possibility that roundoff errors will lead to spurious "FALSE" answers, so the values of (x_3, y_3) are set even when FALSE is returned.*

```
procedure Circle_Crossover($x_1$,$y_1$,$x_2$,$y_2$,$\delta$)
    $a := \delta/2$
    $hip := (x_1 \cdot x_2 + y_1 \cdot y_2)/2$
    if $hip > 0.5$                    "two checks to avoid roundoff errors"
        $hip := 0.5$
    else if $hip < -0.5$
        $hip := -0.5$
    $csqr := .5 - hip$
    $c := \sqrt{csqr}$
    $d := \sqrt{.5 + hip}$
    if $-x_2 \cdot y_1 + x_1 \cdot y_2 < 0$
        $d := -d$
    $asqr := a \cdot a$
    if $asqr > csqr$
        if $a < 0$
            $x_3 := x_1$
            $y_3 := y_1$
        else
            $x_3 := x_2$
            $y_3 := y_2$
        return(FALSE)
    $u := -(1 + d)(1 - (asqr/csqr))$
    if $asqr = csqr$
        $v := a$
    else
        $v := a\sqrt{1 + (u^2)/(csqr - asqr)}$
    if $hip > 0$
        $\alpha := (x_1 + x_2)/(2d)$
        $\beta := (y_1 + y_2)/(2d)$
    else
        $\alpha := (y_2 - y_1)/(2c)$
        $\beta := (x_1 - x_2)/(2c)$
    $x_3 := (u + d)\alpha - v\beta$
    $y_3 := v\alpha + (u + d)\beta$
    return (TRUE)
```

Of course, the crucial implementation difficulty for the procedure Ω is the algorithm for Circle_Crossover; this is shown as Algorithm 3.4. Circle_Crossover takes two points (x_1, y_1) and (x_2, y_2) lying on the unit circle in the xy-plane and a real value δ. The theoretical δ-crossover point of (x_1, y_1) and (x_2, y_2) is found as an intersection point of the unit circle and the hyperbola consisting of those points which have distance from (x_1, y_1) equal to δ plus their distance from (x_2, y_2) (namely, the intersection which is not between (x_1, y_1) and (x_2, y_2) in tour order). Letting the "half inner product" *hip* equal $(x_1x_2+y_1y_2)/2$, the distance between the two points is equal to $2c$, where $c = \sqrt{\frac{1}{2} - hip}$. And, the midpoint of the line segment between the two points is distance $\sqrt{\frac{1}{2} + hip}$ from the origin. To conveniently express the equation for the hyperbola, we set up uv-axes as a rigid translation of the xy-axes, positioned so that the points (x_1, y_1) and (x_2, y_2) have uv-coordinates $(0, -c)$ and $(0, c)$, respectively. This makes the origin have uv-coordinates $(d, 0)$, where $d = \pm\sqrt{\frac{1}{2} + hip}$ with the sign being $+$ if and only if the angle from (x_1, y_1) to (x_2, y_2) is $\leq 180\,\mathrm{deg}$. In the uv-plane, the hyperbola has equation

$$\frac{v^2}{a^2} - \frac{u^2}{c^2 - a^2} = 1$$

where $a = \delta/2$, and the unit circle has equation

$$(u + d)^2 + v^2 = 1.$$

Eliminating v^2 from these equations and solving for u, and then for v, shows that the desired intersection point of the circle and the hyperbola has uv-coordinates

$$u = (1 - d)\left(1 - \frac{a^2}{c^2}\right),$$
$$v = a\sqrt{1 + \frac{u^2}{c^2 - a^2}}.$$

Given the uv-coordinates, it is an easy matter to find values α, β which allow the corresponding xy-coordinates to be computed. Algorithm 3.4 show two equivalent calculations of α, β; the algorithm chooses the one which avoids division by zero or division by a number close to zero. Algorithm 3.4, as shown, also checks for some error conditions that can arise from roundoff errors. In particular, it makes sure that $|hip| \leq \frac{1}{2}$, and that $a^2 < c^2$. Amazingly enough, we found, during extensive testing with randomly generated tours of points on the unit circle, that roundoff error occasionally caused these conditions to be violated, even for points on the unit circle and for $|a| < |c|$.

3.4. An ANSI-C implementation. An efficient and highly portable ANSI-C implementation of our algorithms is described in [5], which includes complete source code, test programs for several interesting cases, benchmark results, and software to produce postscript graphical representations of the matchings found. To help ensure the correctness of our implementation, a straightforward $O(n^3)$ dynamic programming solution was also implemented, and the results compared for 4,000,000 pseudo-randomly drawn problems. Figure 1.1 shows an example of a matching produced by our software.

Benchmark results for a variety of RISC processors produced nearly identical results when normalized by clock rate. So timing results in [5] are given in units of RISC cycles. Graphs of up to $20,000$ nodes are included in this study.

Recall that $O(\log N)$ time is a worst case bound for generic Ω. One interesting experimental result is that over the range of graph sizes considered, for the specific settings implemented in the test programs, and given the uniform pseudorandom manner in which problem instances were generated, the generic Ω implementation exhibits very nearly linear time performance. In other words, the experimentally observed runtime of Ω was nearly constant. We suspect that this is primarily a consequence of the uniform random distribution from which problems were drawn, and that it should be possible to demonstrate expected time results better than $O(\log N)$ for more structured settings.

The benchmarks included one line-like and two circular settings. Solving pseudorandomly drawn matching problems of size n required on average between 2,000 and 16,000 RISC cycles per node depending on the setting and on whether a constant-time or generic Ω was employed. It is interesting to note that, in all cases, the constant-time Ω performed better by a factor ranging from roughly 1.5 to slightly over 3. Thus, for some problems, the linear time result of this paper may be of practical interest.

Despite our focus on efficiency, further code improvements and cost function evaluation by table lookup may contribute to significant performance improvement.

4. Nonbipartite, quasi-convex tours.

In this section we show how the earlier algorithms can be applied to nonbipartite, quasi-convex tours. The principal observation is that nonbipartite tours may be made bipartite by the simple construction of making the nodes alternate in color. This is already observed by Marcotte and Suri [20] in a more restrictive setting; we repeat the construction here for the sake of completeness.

First, it is apparent that the proof of Lemma 2.3 still works in the nonbipartite case, and thus any nonbipartite, quasi-convex tour has a minimum-cost matching in which no jumpers cross. This fact implies the following two lemmas.

LEMMA 4.1. *Let $x_1, \ldots, x_N$ be a nonbipartite, quasi-convex tour with N even. Then there exists a minimum-cost matching such that every edge in the tour is of the form $x_i \leftrightarrow x_j$ with i even and j odd.*

Proof. It will suffice to show that any crossing-free matching has this property. Suppose $x_i \leftrightarrow x_j$ is a jumper in a crossing-free matching, with $i < j$. Since N is even, the matching is complete in that every node is matched. The crossing-free property thus implies that the nodes in (x_i, x_j) are matched with each other, so there are an even number of such nodes, i.e., one of i and j is even and the other is odd. □

LEMMA 4.2. *Let $x_1, \ldots, x_N$ be a nonbipartite line-like quasi-convex tour. Then there exists a minimum-cost matching such that every edge in the tour is of the form $x_i \leftrightarrow x_j$ with i even and j odd.*

Proof. If N is even then this lemma is just a special case of the former lemma. If N is odd, then add an additional node x_{N+1} to the end of the tour, with $c(x_i, x_{N+1}) = 0$ for all i. The resulting tour is again quasi-convex and of even length; so the lemma again follows immediately from the former lemma. □

When Lemmas 4.1 and 4.2 apply, we may color the even nodes red and the odd nodes blue and reduce the nonbipartite matching problem to a bipartite matching problem. As an immediate corollary, we have that the two main theorems also apply in the nonbipartite setting; namely, for nonbipartite, quasi-convex tours of even length and for nonbipartite, line-like, quasi-convex tours, the matching problem can always be solved in $O(N \log N)$ time and it can be solved in $O(N)$ time if the weak analyticity condition holds.

We do not know whether similar algorithms exist for the case of general (i.e., nonline-like) quasi-convex tours of odd length. Similarly, we do not know any linear or near-linear time algorithms for bipartite, quasi-convex tours which are neither balanced nor line-like.

We conclude this section by mentioning a tantalizing connection between our work and the work of Yao [25]. Yao gave a quadratic runtime algorithm for solving the dynamic programming problem

$$d(i,j) \;=\; c(i,j) + \min\{d(i,k-1) + d(k,j) : i < k \le j\}$$

for line-like quasi-convex tours with cost function c (improving on the obvious cubic-time algorithm). Our nonbipartite matching problem can be stated as a similar dynamic programming problem; namely, the minimum-cost, $MC(i,j)$, of a complete matching on the nodes in $[x_i, x_j]$ can be recursively defined to equal

$$\min\{c(i,k) + MC(i+1,k-1) + MC(k+1,j) : i < k \le j\}.$$

(A similar dynamic programming algorithm can be given for the bipartite matching problem.) The obvious naive algorithm for computing $MC(-,-)$ is cubic-time; however, our main results give (near)-linear time algorithms for line-like quasi-convex tours. This raises the possibility that the dynamic programming problem considered by Yao may also have a near-linear time solution.

5. Applications to string matching. As a final topic we briefly discuss the application of our matching results to string comparison. A full treatment is beyond the scope of this paper, but additional details and related algorithms may be found in [6]. Given two symbol strings $v = a_1a_2\cdots a_n$ and $w = b_1b_2\cdots b_n$, our goal is to measure a particular notion of *distance* between them. Intuitively, distance acts as a measure of similarity; i.e., strings that are highly similar (highly dissimilar) are to have a small (large) distance between them. The purpose of such formulations is usually to approximate human similarity judgments within a pattern classification or information retrieval system.

Suppose $f(x)$ is a monotonely increasing, concave-down function with $f(0) = 0$. Let symbols $a_1, \ldots, a_n$ in v be a graph's red nodes and $b_1, \ldots, b_n$ in w be its blue nodes, and consider bipartite matchings of these $2n$ symbols. In the simplest formulation, we define the cost of an edge $a_i \leftrightarrow b_j$ as $f(|j-i|)$ if a_i and b_j are the same symbol and as $f(n)$ if a_i and b_j are distinct symbols. The cost of matching unequal characters can also be set to be any other fixed value instead of $f(n)$. Our *distance*, $\sigma(v,w)$, between strings v and w is then the minimum cost of any such bipartite matching.

As an example, consider the two strings "delve" and "level" and let $f(x) = \sqrt{x}$. Then the distance between these two strings is $\sqrt{5} + \sqrt{0} + \sqrt{2} + \sqrt{1} + \sqrt{1} \approx 5.65$.

As we have set up our problem above, the computation of $\sigma(v,w)$ is not directly an instance of the quasi-convex matching problem. However we can compute the σ function by considering each alphabet symbol α separately, and solving the quasi-convex matching problem σ_α which results from restricting attention to occurrences of a single alphabet symbol at a time. To make this clear, we introduce a special symbol "-" which indicates the absence of an alphabet symbol. The value of σ("delve," "level") can be expressed as the sum

$$\begin{aligned}&\sigma_d(\text{“}d\text{----”}, \text{“-----”}) + \sigma_e(\text{“-}e\text{--}e\text{”}, \text{“-}e\text{-}e\text{-”})\\ &\quad + \sigma_l(\text{“--}l\text{--”}, \text{“}l\text{---}l\text{”}) + \sigma_v(\text{“---}v\text{-”}, \text{“--}v\text{--”}).\end{aligned}$$

To make the summed σ_α terms equal σ as originally defined, each σ_α is defined to be the subproblem's minimum matching cost plus $f(n)/2$ times the number of unmatched symbols.

We will loosely refer to distance functions that result from this kind of formulation as σ-distances. Assuming that $f(x)$ satisfies the weak analyticity condition, it is not too difficult to show that it is possible to compute $\sigma(v, w)$ in linear time. If the weak analyticity condition does not hold, then our results give an $O(n \log n)$ time algorithm.

A novel feature of our σ-distances is that distinct alphabet symbols are treated independently. This is in contrast to most prior work which has used "least edit distance" for string comparison (see [21] for a survey). As an illustration of the difference between our distance measure and the "edit distance" approach, consider comparing the word "abcde" with its mirror image "edcba." Our approach recognizes some similarity between these two forms, while the most standard "edit distance" approach sees only that the two strings have "c" in common—in essence substituting the first two and last two symbols of the string without noticing the additional occurrences of the same symbols at the other end of the other string.

A special form of our σ-distance measure in which $f(x) = x$, and the optimal matching is only approximated, was introduced earlier by the authors and shown to have a simple linear time algorithm [26, 27]. Its relationship to σ-distances is described in [6]. This earlier algorithm has been successfully used in commercial applications, especially for spelling correction in word processing software, typewriters, and hand-held dictionary devices (we estimate that over 15,000,000 such software/hardware units have been sold by Proximity Technology, Franklin Electronic Publishers, and their licensees). Other less prominent commercial applications include database field search (e.g., looking up a name or address), and the analysis of multifield records such as mailing addresses, in order to eliminate near-duplicates. In both of these applications, the strict global left-right ordering imposed by $O(n^2)$ time "edit distance" methods can be problematic. On the other hand, very local left-right order preservation seems to be an important part of similarity perception in humans. One simple adaptation of our σ-distance methods which goes a long way toward capturing this characteristic consists of extending the alphabet beyond single symbols to include digraphs or multigraphs. The result is increased sensitivity to local permutation. Another effective alphabet extension technique involves the addition of *feature symbols* to the alphabet to mark events such as likely phonetic transitions. We expect that the use of general concave-down distance functions (as opposed to $f(x) = x$) will improve the quality of the similarity judgments possible within the σ-distance framework.

The development above considers strings of equal length only. The unequal length case is not a difficult generalization but considering it does highlight the issue of *embedding*. By this we mean that it is implicit in our formulation that the two strings are in a sense embedded into the real line. The particular, rather natural embedding we've assumed so far maps a_i and b_i to value i on the real line, but others are possible.

A detailed comparison of our methods with "edit distance" approaches is beyond the scope of this paper. But we must point out that the "edit distance" formulation is in several senses richer than ours. First, the cost of matching different alphabet members need not be fixed. Also, our distance formulation depends on a designated embedding while the "edit distance" method requires no such specification. Finally, for some problems left-right order preservation may be desirable. On the other hand, even the simplest "edit distance" approach is $O(n^2)$ compared with the $O(n)$ or $O(n \log n)$ complexity of our method. We therefore feel that additional work is needed to better understand the applications of our approach—and perhaps extend it.

Acknowledgments. We wish to thank Dina Kravets, Dave Robinson, and Warren Smith for helpful discussions and Dave Robinson and Kirk Kanzelberger for implementing and testing the algorithms described above.

REFERENCES

[1] A. Aggarwal, A. Bar-Noy, S. Khuller, D. Kravets, and B. Schieber, *Efficient minimum cost matching using quadrangle inequality*, in Proc. 33rd Annual IEEE Symp. on Foundations of Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 1992, pp. 583–592.

[2] A. Aggarwal and M. Klawe, *Applications of generalized matrix searching to geometric algorithms*, Discrete Appl. Math., 27 (1990), pp. 3–23.

[3] A. Aggarwal, M. Klawe, S. Moran, P. Shor, and R. Wilber, *Geometric applications of a matrix-searching algorithm*, Algorithmica, 2 (1987), pp. 195–208.

[4] A. Aggarwal and J. Park, *Notes on searching in multidimensional monotone arrays*, in Proc. 29th Annual IEEE Symposium on Foundations of Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 1988, pp. 497–512.

[5] S. R. Buss, K. G. Kanzelberger, D. Robinson, and P. N. Yianilos, *Solving the Minimum-cost Matching Problem for Quasi-convex Tours: An Efficient ANSI-C Implementation*, Tech. Report CS94-370, University of California, San Diego, 1994.

[6] S. R. Buss and P. N. Yianilos, *A Bipartite Matching Approach to Approximate String Comparison and Search*, Tech. Report, NEC Research Institute, Princeton, NJ, 1995.

[7] D. Eppstein, *Sequence comparison with mixed convex and concave costs*, J. Algorithms, 11 (1990), pp. 85–101.

[8] Z. Galil and R. Giancarlo, *Speeding up dynamic programming with applications to molecular biology*, Theoret. Comput. Sci., 64 (1989), pp. 107–118.

[9] Z. Galil and K. Park, *A linear-time algorithm for concave one-dimensional dynamic programming*, Inform. Process. Lett., 33 (1990), pp. 309–311.

[10] P. Gilmore and R. Gomory, *Sequencing a one state-variable machine: A solvable case of the traveling salesman problem*, Oper. Res., 12 (1964), pp. 655–679.

[11] X. He, *An efficient parallel algorithm for finding minimum weight matching for points on a convex polygon*, Inform. Process. Lett., 37 (1991), pp. 111–116.

[12] D. S. Hirschberg and L. L. Larmore, *The least weight subsequence problem*, SIAM J. Comput., 16 (1987), pp. 628–638.

[13] A. J. Hoffman, *On simple linear programming problems*, in Convexity: Proceedings of the Seventh Symposium in Pure Mathematics of the AMS, V. Klee, ed., American Mathematical Society, Providence, RI, 1963, pp. 317–327.

[14] R. M. Karp and S.-Y. R. Li, *Two special cases of the assignment problem*, Discrete Math., 13 (1975), pp. 129–142.

[15] M. M. Klawe and D. J. Kleitman, *An almost linear time algorithm for generalized matrix searching*, SIAM J. Discrete Math., 3 (1990), pp. 81–97.

[16] D. Kravets and J. K. Park, *Selection and sorting in totally monotone arrays*, Math. Systems Theory, 24 (1991), pp. 201–220.

[17] L. L. Larmore and B. Schieber, *On-line dynamic programming with applications to the prediction of RNA secondary structure*, J. Algorithms, 12 (1991), pp. 490–515.

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

[19] Y. Mansour, J. K. Park, B. Schieber, and S. Sen, *Improved selection in totally monotone arrays*, Internat. J. Comput. Geom. Appl., 3 (1993), pp. 115–132.

[20] O. Marcotte and S. Suri, *Fast matching algorithms for points on a polygon*, SIAM J. Comput., 20 (1991), pp. 405–422.

[21] D. Sankoff and J. B. Kruskal, *Time Warps, String Edits and Macromolecules: The Theory and Practice of Sequence Comparison*, Addison–Wesley, Reading, MA, 1983.

[22] P. M. Vaidya, *Geometry helps in matching*, SIAM J. Comput., 18 (1989), pp. 1201–1225.

[23] M. Werman, S. Peleg, R. Melter, and T. Kong, *Bipartite graph matching for points on a line or a circle*, J. Algorithms, 7 (1986), pp. 277–284.

[24] R. Wilber, *The concave least-weight subsequence problem revisited*, J. Algorithms, 9 (1988), pp. 418–425.

[25] F. F. Yao, *Speed-up in dynamic programming*, SIAM J. Alg. Discrete Methods, 3 (1982), pp. 523–540.

[26] P. N. Yianilos, *The Definition, Computation and Application of Symbol String Similarity Functions*, Master's thesis, Emory University, Atlanta, GA, 1978.

[27] P. N. Yianilos and S. R. Buss, *Associative memory circuit system and method, continuation-in-part*, U.S. Patent 4490811, 1984.