1996

Applications of a Numbering Scheme for Polygonal Obstacles in the Plane

Mikhail J. Atallah
Purdue University, mja@cs.purdue.edu

Danny Z. Chen

Report Number:
96-055

APPLICATIONS OF A NUMBERING SCHEME FOR POLYGONAL OBSTACLES IN THE PLANE

Mikhail J. Atallah
Danny Z. Chen

CSD-TR 96-055
September 1996

Applications of a Numbering Scheme for Polygonal Obstacles in the Plane*

Mikhail J. Atallah[†] Danny Z. Chen[‡]

Abstract

We present efficient algorithms for the problems of matching red and blue disjoint geometric obstacles in the plane and connecting the matched obstacle pairs with mutually nonintersecting paths that have useful geometric properties. We first consider matching n red and n blue disjoint isothetic rectangles and connecting the n matched rectangle pairs with nonintersecting monotone rectilinear paths; each such path consists of $O(n)$ segments and is not allowed to touch any rectangle other than the matched pair that it is linking. Based on a numbering scheme for certain geometric objects and on several useful geometric observations, we develop an $O(n \log n)$ time, $O(n)$ space algorithm that produces a desired matching for isothetic rectangles. If an explicit printing of all the n paths is required, then our algorithm takes $O(n \log n + \lambda)$ time and $O(n)$ space, where λ is the total size of the desired output. We then extend these matching algorithms to other classes of red/blue polygonal obstacles. The numbering scheme also finds applications to other problems.

1 Introduction

The problem of computing paths that avoid obstacles is fundamental in computational geometry and has many applications. It has been studied in both sequential and parallel settings and using various metrics. The rectilinear version of the problem, which assumes that each of a path's constituent segments is parallel to a coordinate axis, is motivated by applications in areas such as VLSI wire layout, circuit design, plant and facility layout, urban transportation, and robot motion. There are many efficient sequential algorithms that compute various shortest rectilinear paths avoiding different classes of obstacles [11, 12, 13, 14, 15, 17, 18, 19, 21, 24, 25, 27, 28, 29, 38, 39, 40, 41], and some parallel algorithms as well [4, 5].

In this paper, we present efficient algorithms for the problems of matching red and blue disjoint geometric obstacles in the plane and connecting the matched obstacle pairs with mutually nonintersecting paths that have certain useful geometric properties. The first problem we consider has the following input: n of the given $2n$ pairwise disjoint isothetic rectangles are

*This work was carried out in part at the Center for Applied Science and Engineering and Institute of Information Science, Academia Sinica, Nankang, Taiwan, during its 1996 Summer Institute on Computational Geometry and Applications.

†Department of Computer Sciences, Purdue University, West Lafayette, Indiana 47907, USA. mja@cs.purdue.edu. This author gratefully acknowledges support from the National Science Foundation under Grant DCR-9202807, and the COAST Project at Purdue University and its sponsors, in particular Hewlett Packard.

‡Department of Computer Science and Engineering, University of Notre Dame, Notre Dame, Indiana 46556, USA. chen@cse.nd.edu. The research of this author was supported in part by the National Science Foundation under Grant CCR-9623585.

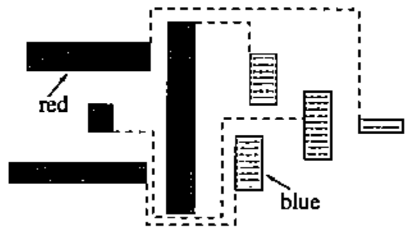

Figure 1: Example of a matching of red and blue rectangles with monotone paths.

colored red (think of them as sources of something, e.g., electric power in a VLSI circuit), and the other n are colored blue (think of them as consumers of power). By *isothetic* objects, we mean that each edge of such an object is parallel to a coordinate axis. We are interested in matching each red rectangle with one and only one blue rectangle, and *vice versa*. Specifically, we would like to find such a matching and connect each matched pair of red/blue rectangles with a planar rectilinear path in such a way that (i) each path is monotone with respect to a coordinate axis, (ii) each path does not touch any rectangle other than the matched pair that it is supposed to connect, (iii) no two such paths intersect each other, and (iv) each path consists of $O(n)$ segments. Figure 1 shows an example of such a matching.

Several geometric algorithms have been developed for solving various problems of finding obstacle-avoiding pairwise disjoint paths that connect certain geometric objects [26, 32, 36], because of their relevance to VLSI layout applications [16, 26, 35] (e.g., VLSI single-layer routing). Lee *et al.* [26] designed an $O((k^2!)n \log n)$ time algorithm for computing k shortest non-crossing rectilinear paths in a *plane region*. Takahashi, Suzuki, and Nishizeki [36] studied the problem of finding shortest non-crossing rectilinear paths in a plane region that is bounded by an outer box and an inner box and that contains a set of disjoint isothetic rectangle obstacles, giving an $O(n \log n)$ time algorithm for computing k such paths whose endpoints are all on the two bounding boxes (with $k \leq n$). Papadopoulou [32] very recently obtained an $O(n + k)$ time algorithm for computing k shortest non-crossing paths in a simple polygon whose endpoints are all on the polygon boundary. However, these problems are different from the one we study here since they often assume that a specification on which object matches with which other object is already given (hence, these problems require only to compute a set of non-crossing paths that realize the specified matching).

We develop an $O(n \log n)$ time, $O(n)$ space algorithm that produces a desired matching for red/blue isothetic rectangles. If an explicit printing of all the n paths for such a matching is required, then our algorithm takes $O(n \log n + \lambda)$ time and $O(n)$ space, where λ is the total size of the desired output.

We then extend these matching algorithms to a more general geometric setting which consists of disjoint red/blue polygonal obstacles that are all monotone to a coordinate axis (say, the y-axis). The matching paths we compute for this more general setting have similar structures to those for isothetic rectangles, except that in this case their monotonicity has to be weaker: Each such matching path can be partitioned into at most two subpaths, each of which is monotone to the y-axis. Our matching algorithms for y-monotone polygonal obstacles have the same complexity bounds as those for isothetic rectangles.

We also prove that all the matching problems studied in this paper have an $\Omega(n \log n)$ lower bound in the algebraic computation tree model [8]. Our matching algorithms are based on a numbering scheme for certain geometric objects and on several useful geometric observations. This numbering scheme also finds applications to other problems [7].

Our algorithms can also be viewed as proofs that such matchings always exist, a fact that, to the best of our knowledge, was not previously established. We should point out that without the requirement that all matching paths must satisfy a monotonicity constraint, the existence of nonintersecting paths for any red/blue polygonal obstacle matching is trivial to prove: For every matched pair of geometric objects in turn, draw a direct rectilinear path P between them, ignoring all previously drawn paths and obstacles; at each place where path P intersects a previously drawn path or an obstacle, "deform" P so that P goes around that previously drawn path or the obstacle.

Section 2 gives some preliminary definitions, Section 3 presents one of the ingredients needed by the matching algorithms for isothetic rectangles, Section 4 describes the data structures that our matching algorithms will use, Section 5 gives the algorithm for computing a desired matching for isothetic rectangles, Section 6 extends this algorithm to also producing the n actual monotone paths that link the matched rectangle pairs, Section 7 generalizes these algorithms to matching y-monotone polygonal obstacles, Section 8 proves $\Omega(n \log n)$ lower bounds for the matching problems we consider, and Section 9 makes further remarks on several consequences and possible extensions of this work.

2 Preliminaries

A geometric object in the plane is *rectilinear* if each of its constituent segments is parallel to either the x-axis or the y-axis. Without loss of generality (WLOG), we assume that no two boundary edges of the input obstacles are collinear. We use $R = \{R_1, R_2, \ldots, R_{2n}\}$ to denote the set of $2n$ input isothetic rectangles.

A path consists of a contiguous sequence of line segments in the plane. The number of line segments in a path P is called the *size* of P, denoted by $|P|$, and the *length* of P is the sum of the distances of its edges in a certain metric. A path is said to be *monotone* with respect to the x-axis (resp., y-axis) if and only if its intersection with every vertical (resp., horizontal) line is either empty or a contiguous portion of that line. A path is said to be *monotone* if and only if it is monotone to the x-axis or to the y-axis. A rectilinear path is *convex* if it is monotone to both the x-axis and the y-axis. In general, a convex (rectilinear) path has the shape of a staircase, and in fact we shall henceforth use the word "staircase" as a shorthand for "convex path". Staircases can be either *increasing* or *decreasing*, depending on whether they go up or down as we move along them from left to right. A staircase is *unbounded* if it starts and ends with a semi-infinite segment, i.e., a segment that extends to infinity on one end. A staircase is said to be *clear* if it does not intersect the interior of any input obstacle.

A polygon G is said to be *monotone* to the x-axis (resp., y-axis) if and only if its intersection with any vertical (resp., horizontal) line L is either empty or a contiguous segment of L; the boundary of such a monotone polygon G can be partitioned into two paths each of which is monotone to the x-axis (resp., y-axis). In fact, the notion of monotonicity of a polygon or a path is in general with respect to an arbitrary line [34]. Note that it is possible to find out in linear time whether there is a line (in an arbitrary direction) to which all polygons in a polygon set are monotone, by using Preparata and Supowit's monotonicity test algorithm [34].

A point p in the plane is defined by its x-coordinate $x(p)$ and y-coordinate $y(p)$. A point p is strictly *below* (resp., to the *left* of) a point q if and only if $x(p) = x(q)$ and $y(p) < y(q)$ (resp., $y(p) = y(q)$ and $x(p) < x(q)$); we can equivalently say that q is strictly *above* (resp., to the *right* of) p. A rectangle r is *below* (resp., to the *left* of) an unbounded staircase S if no point of r is strictly above (resp., to the right of) a point of S; we can equivalently say that S is *above* (resp., to the *right* of) r.

Unless otherwise specified, all geometric objects in the rest of this paper (e.g., paths, rays, lines, polygons, obstacles, etc) are assumed to be rectilinear in the plane.

3 Partitioning Isothetic Rectangles with a Staircase

Given a set $R = \{R_1, R_2, \ldots, R_{2n}\}$ of $2n$ pairwise disjoint isothetic rectangles in the plane and an integer k with $1 \leq k < 2n$, we present an algorithm for partitioning the set R into two subsets of respective sizes k and $2n - k$, such that the two resulting subsets are separated by an increasing staircase. This algorithm runs in $O(n \log n)$ time, or in $O(\min\{k, 2n-k\})$ time if R is given in a suitably preprocessed form. The algorithm can also be implemented optimally in parallel (see Section 9). A key idea of this partition algorithm is a useful numbering scheme for certain geometric objects, which also finds applications to other problems [7].

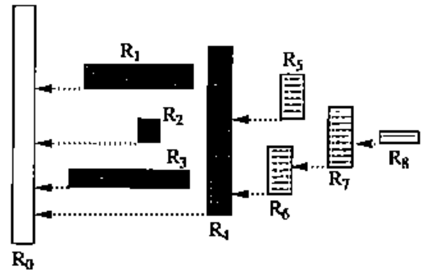

Figure 2: Illustrating the tree T of the rectangles in R.

Not only is the result of this section needed as a key ingredient in the algorithms for matching isothetic rectangles given later, but it also implies simpler algorithms for a number of unrelated divide-and-conquer sequential and parallel algorithms for various rectilinear shortest path problems among rectangles, in which such a staircase is needed for bipartitioning the problem before recursively solving the two subproblems defined by the staircase [4, 5, 11, 29].

We begin by describing the $O(n \log n)$ time preprocessing. The first step of the preprocessing algorithm consists of computing a horizontal trapezoidal decomposition of R [33], in $O(n \log n)$ time. Recall that this gives, among other things, the following *Parent* information (actually, it gives more than what follows, but we only need what follows): For each rectangle R_i of R, *Parent*(i) is the first rectangle R_j encountered by shooting a leftwards-moving horizontal ray from the *bottom-left corner* of R_i (see Figure 2). If no such rectangle exists for R_i, then the ray goes to infinity, a fact that we denote by saying that *Parent*(i) is empty. Note that the rectangles in R and their *Parent* information together define a forest of the rectangles. The trapezoidal decomposition algorithm [33] also produces a sorted list of each subset of rectangles having the same *Parent* (including the "empty" parent). Every rectangle R_j maintains an adjacency list of all the rectangles whose *Parent* is R_j, sorted by the *decreasing* y-coordinates of their leftwards-moving horizontal rays. For example, the sorted adjacency list of R_4 in Figure 2 is $\{R_5, R_6\}$.

The second step of the preprocessing algorithm is now given. To simplify the presentation, we assume that we have added to the given collection R of input rectangles an extra "dummy" rectangle R_0 to the left of all the other rectangles in R, such that the horizontal projection of R_0 on the y-axis properly contains the horizontal projections of all the other rectangles (see Figure 2). This amounts to replacing every empty *Parent*(i) by R_0, effectively making R_0 the root of a tree each of whose nodes corresponds to exactly one rectangle in R. We use T to denote this tree. Figure 2 shows an example of such a tree T. The preprocessing algorithm then computes the preorder numbers of T in $O(n)$ time [1], and re-labels the rectangles of R (which are the nodes of T) so that rectangle R_i now denotes the one whose preorder number in T is i. The

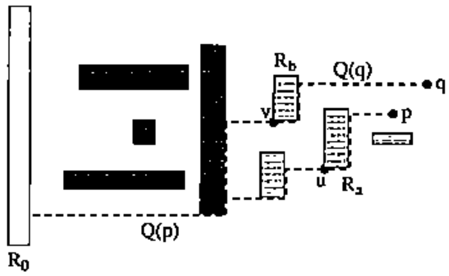

Figure 3: An example of the paths $Q(p)$ and $Q(q)$.

preorder numbers of T start from 0. Hence the dummy rectangle, the root, retains the name R_0. This completes the description of the preprocessing.

This preprocessing algorithm clearly takes altogether $O(n \log n)$ time and $O(n)$ space. In the rest of this section, we assume that the rectangles have been re-labeled as explained above.

For any set R' of disjoint rectangles, we henceforth use $CH(R')$ to denote the rectilinear convex hull of R' in the plane (see [30] for a study of rectilinear convex hulls of planar rectilinear geometric objects). For every point p in the plane that is to the right of the root rectangle R_0 and is not in the interior of any obstacle, we define a path $Q(p)$ from p to R_0, as follows:

> $Q(p)$ starts at p and follows the leftwards-moving horizontal ray $r(p)$ from p; if the ray $r(p)$ first hits a rectangle $R_i \neq R_0$, then $Q(p)$ goes downwards along the boundary of R_i to its bottom-right vertex and then leftwards to its bottom-left vertex, from which $Q(p)$ continues as it did at p, until it reaches R_0.

Note that for every such point p, the path $Q(p)$ is uniquely defined, and in fact is always an increasing obstacle-avoiding staircase. Also, note that every vertical segment of $Q(p)$ is completely on the right edge of a rectangle and the lower vertex of such a vertical segment is at the bottom-right vertex of that rectangle. Figure 3 gives an example of such paths.

The following lemmas are useful to proving the theorem on staircase separators.

Lemma 1 *Let p and q be two points in the plane such that they both are to the right of R_0, and $x(p) \leq x(q)$. If p is below (resp., above) some point of $Q(q)$, then no point of $Q(p)$ is strictly above (resp., below) any point of $Q(q)$ (Figure 3).*

Proof. The proof is straightforward and will be given in the full version of the paper. □

Lemma 2 *Let p and q be two points in the plane such that they both are to the right of R_0 and that $x(p) \leq x(q)$. Let u (resp., v) be the bottom-left vertex of a rectangle R_a (resp., R_b), such that u (resp., v) is on $Q(p)$ (resp., $Q(q)$) but not on $Q(q)$ (resp., $Q(p)$). If p is strictly below (resp., above) some point of $Q(q)$, then the preorder number of R_a in the tree T of rectangles is larger (resp., smaller) than that of R_b, i.e., $a > b$ (resp., $a < b$).*

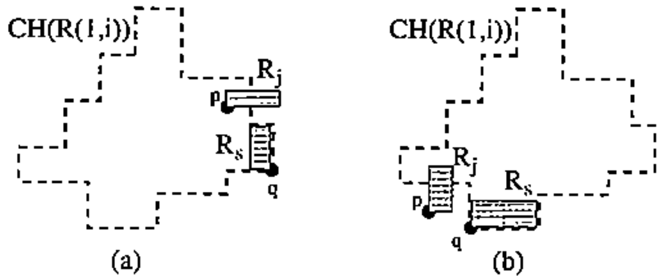

Figure 4: Illustrating the proof of the staircase separator theorem.

Proof. This follows from Lemma 1 and from the definition of the tree T. An example illustrating the lemma is given in Figure 3. □

We are now ready to give the staircase separator theorem.

Theorem 1 (Staircase Separator Theorem) *Given a preprocessed set R of $2n$ disjoint isothetic rectangles, the subsets $\{R_1, R_2, \ldots, R_k\}$ and $\{R_{k+1}, R_{k+2}, \ldots, R_{2n}\}$, for any integer k with $1 \leq k < 2n$, form a partition of the set R that has the desired property, that is, there exists a rectangle-avoiding increasing staircase of size $O(n)$ that separates these two subsets. Furthermore, such a staircase separator can be computed in $O(\min\{k, 2n-k\})$ time.*

Proof. Let $R(a,b)$ denote the subset $\{R_a, R_{a+1}, \ldots, R_b\}$ of R. For the existence of such a staircase separator, we first show that for any $i < j$, the following holds: (1) $CH(R(1,i))$ does not intersect R_j, and (2) $CH(R(j,2n))$ does not intersect R_i. We give the proof only for (1), that for (2) being similar. We prove (1) by contradiction: Suppose to the contrary that for some $j > i$, R_j intersects $CH(R(1,i))$. Then one of the following two cases must occur:

- $CH(R(1,i))$ contains some point p on the bottom edge of R_j ($CH(R(1,i))$ possibly contains R_j completely). Note that there can be no rectangles R_s and R_l, $s \leq i < l$, such that the leftwards-moving horizontal ray from the bottom-left vertex of R_s first hits R_l (otherwise, this would make R_l the parent of R_s, contradicting the fact that R_l has a larger preorder number than R_s in the tree T). Since the point p of R_j is inside $CH(R(1,i))$, there must be a rectangle R_s, $s \leq i < j$, such that the bottom edge of R_s contains a point q that satisfies both $x(p) \leq x(q)$ and $y(p) > y(q)$ (see Figure 4(a)). But then the path $Q(p)$ (resp., $Q(q)$) contains the bottom-left vertex of R_j (resp., R_s) and by Lemma 2, the preorder number of R_j in T is smaller than that of R_s, a contradiction.

- $CH(R(1,i))$ contains some point of R_j but the bottom edge of R_j is completely outside $CH(R(1,i))$. Then R_j must intersect the lower hull of $CH(R(1,i))$ (see Figure 4(b)). Again there can be no rectangles R_s and R_l, $s \leq i < l$, such that the leftwards-moving horizontal ray from the bottom-left vertex of R_s first hits R_l. But then, there must be a point q on the bottom edge of a rectangle R_s, $s \leq i < j$, such that $x(p) \leq x(q)$ and $y(p) > y(q)$ for some point p on the bottom edge of R_j (Figure 4(b)). Again by Lemma 2, this implies that the preorder number of R_j in T is smaller than that of R_s, a contradiction.

We can now let such a desired staircase separator S for the subsets $R(1,k)$ and $R(k+1,2n)$ consist of (say) the portion of the boundary of $CH(R(1,k))$ from its rightmost edge clockwise to its lowest edge, augmented by two semi-infinite segments, one extended leftwards horizontally from its lowest edge and the other extended upwards vertically from its rightmost edge. By using the same arguments as above, we can show that for every j with $j > k$, S is above or to the left of R_j. Hence S so constructed is an obstacle-avoiding increasing staircase and consists of $O(k)$ segments.

WLOG, assume $k = \min\{k, 2n-k\}$. We now show how to compute such a staircase separator S in $O(k)$ time. In fact, we will compute $CH(R(1,k))$, which is a little more than the above staircase S, in $O(k)$ time. Note that the boundary of $CH(R(1,k))$ can be obtained from four staircase paths, each of which corresponds to an ordered sequence of certain suitably defined elements of maximal domination [33] for the $4k$ rectangle vertices of $R(1,k)$. WLOG, we only show the procedure for computing one such sequence of maximal elements.

Our procedure is based on a simple divide-and-conquer strategy. First, partition the set $R(1,k)$ into two subsets $R(1,k/2)$ and $R(k/2,k)$ (WLOG, assume k is an even integer greater than 1). Then, recursively compute the sequence of maximal elements for each such subset, represented by a balanced search tree, such as a 2-3 tree [1]. Finally, compute the sequence of maximal elements for the vertices of $R(1,k)$ from the two sequences for the two subsets. By the above discussion, these two sequences are respectively contiguous portions of the boundaries of two disjoint convex hulls. Hence by performing $O(1)$ standard 2-3 tree operations, the sequence of maximal elements for $R(1,k)$ can be obtained, also maintained by a 2-3 tree. The recurrence relation for the time complexity of this divide-and-conquer procedure is

$T(k) = 2T(k/2) + O(\log k)$, for $k > 1$

$T(1) = O(1)$

Hence it follows that $T(k) = O(k)$. After the above divide-and-conquer procedure terminates, it is easy to obtain the sequence of maximal elements for $R(1,k)$ from its 2-3 tree in $O(k)$ time. The space used for computing $CH(R(1,k))$ is clearly $O(k)$.

This completes the proof of the staircase separator theorem. □

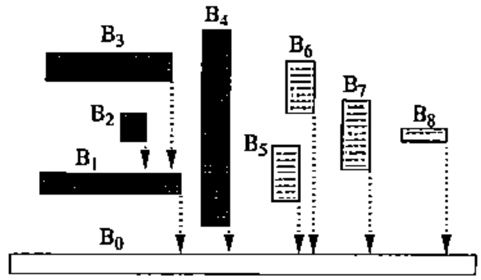

Figure 5: Illustrating the definition of the tree T'.

4 Data Structures

In this section, we describe the data structures that the algorithm in the next section will use. Since that algorithm from time to time will delete some rectangles from the collection $R = \{R_1, R_2, \ldots, R_{2n}\}$, we use L_+ to denote the current list of rectangles sorted by their preorder numbers in T. The list L_+ is initially $\{R_1, R_2, \ldots, R_{2n}\}$, but may change as the algorithm proceeds. However, the following invariants must hold:

1. The list L_+ must contain as many red as blue rectangles.

2. $CH(L_+)$ does not intersect any of the rectangles in $R - L_+$. This invariant insures that we can solve the problem on L_+ without having to worry about interfering with the solution of $R - L_+$, so long as our solution paths for L_+ (resp., $R - L_+$) do not wander outside (resp., inside) of $CH(L_+)$. Note that if the algorithm decides to match the pair of rectangles R', R'' and delete R', R'' from L_+, then this invariant requires that the resulting new list $L_+ - \{R', R''\}$ should also satisfy the invariant, i.e., that $CH(L_+ - \{R', R''\})$ must intersect neither R' nor R''.

We define another list L_- which contains exactly the same set of rectangles as L_+ but is ordered differently (as explained next). L_- initially contains all the input rectangles, but they are sorted according to their preorder numbers in a tree T' rather than T, where T' is defined just like T except for the following differences:

- Instead of the "leftwards-shooting horizontal ray emanating from the bottom-left corner of each rectangle" that we used in the definition of T, in T' we use "downwards-shooting vertical ray emanating from the bottom-right corner of each rectangle" (see Figure 5).

- Instead of sorting adjacency lists by the decreasing y-coordinates of the horizontal shooting rays, in T' the adjacency lists are sorted by the *increasing* x-coordinates of the vertical shooting rays.

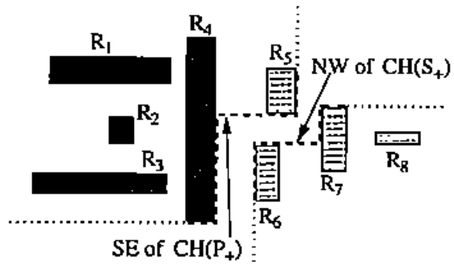

Figure 6: An example for Lemma 3, with $P_+ = \{R_1, \ldots, R_5\}$ and $S_+ = \{R_6, R_7, R_8\}$.

- The "dummy" rectangle corresponding to the root of T' is *below* all the input rectangles (whereas for T it was to their left).

Figure 5 illustrates the tree T' in which the rectangles are named B_i's (for boxes) instead of R_i's.

The L_- list is not explicitly maintained by our algorithm. But, the order in which the elements of L_+ would appear in this hypothetical list L_- is conceptually important, and will be exploited by our algorithm; we henceforth use the shorthand "T' preorder" to refer to this order.

Because L_+ (hence L_-) satisfies Invariant 2 above, the proofs of the following lemmas are very similar to the proof of Theorem 1 and are therefore omitted. (Note how the proof falls apart without Invariant 2, specifically at the place where we deduce that R_l must be the parent of R_s — this need not hold if Invariant 2 is violated, and indeed we cannot even claim that R_l is an ancestor of R_s.)

Lemma 3 *Let P_+ be a prefix of the list L_+, and S_+ be the remaining suffix of L_+, i.e., $S_+ = L_+ - P_+$. Then the increasing staircase defined by the South-East portion of $CH(P_+)$ is (geometrically) above all of the rectangles in S_+. Equivalently, the increasing staircase defined by the North-West portion of $CH(S_+)$ is below all of the rectangles in P_+.*

Figure 6 illustrates Lemma 3.

Lemma 4 *Let P_- be a prefix of the list L_-, and S_- be the remaining suffix of L_-, i.e., $S_- = L_- - P_-$. Then the decreasing staircase defined by the North-East portion of $CH(P_-)$ is (geometrically) below all of the rectangles in S_-. Equivalently, the decreasing staircase defined by the South-West portion of $CH(S_-)$ is above all of the rectangles in P_-.*

Figure 7 illustrates Lemma 4.

When the algorithm to be described in the next section is solving a problem corresponding to the rectangles in L_+, it is not given just the list L_+ but rather a tree structure $S(L_+)$ built

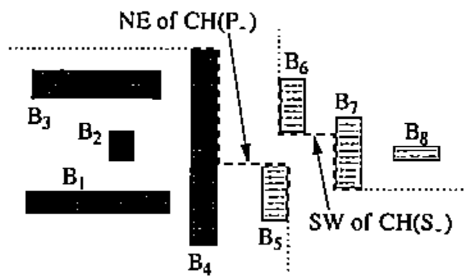

Figure 7: An example for Lemma 4, with $P_- = \{B_1, \ldots, B_5\}$ and $S_- = \{B_6, B_7, B_8\}$.

"on top" of L_+. Specifically, $S(L_+)$ is a 2-3 tree structure [1] whose leaves contain the rectangles in L_+, in the same order as in L_+; these leaves are doubly linked together. Each internal node v of $S(L_+)$ contains a *label* equal to the smallest T' preorder number (i.e., according to the L_- ordering) of the rectangles stored in the subtree of $S(L_+)$ rooted at v. In addition, there are cross-links between every internal node v of $S(L_+)$ and the leaf in the subtree of $S(L_+)$ rooted at v corresponding to the label of v. We will perform only deletion and split operations on $S(L_+)$, both of which can be done in logarithmic time using standard techniques [1]. The deletions will take place after we have matched a pair of rectangles — we then delete them from $S(L_+)$ and recurse on the resulting $S(L_+)$. The split operations will take place when we process L_+ by solving recursively two pieces of L_+: A prefix L' of L_+, and the remaining suffix $L'' = L_+ - L'$ (of course L' and L'' must satisfy the required invariants mentioned earlier). Splitting $S(L_+)$ allows us to create $S(L')$ and $S(L'')$ in logarithmic time.

5 The Matching Algorithm for Rectangles

The goal of this procedure is to compute a desired matching for the rectangles in R, without worrying about describing the actual paths that join the matched pairs of red/blue rectangles (the next section explains how this procedure can be modified to also produce the actual paths connecting the matched pairs).

The procedure is recursive, and takes as input the 2-3 tree data structure $S(L_+)$ described in the previous section.

Procedure MATCH(L_+)

Input: $S(L_+)$, where $L_+ = (R'_1, R'_2, \ldots, R'_m)$.

Output: A matching of the red and blue rectangles in L_+.

1. If $m = 2$, then the two rectangles in L_+ surely have different colors (by Invariant 1): Match them and return. If $m > 2$, then proceed to the next step.

   Comment: The path that will join the pair just matched will be along the boundary of

$CH(L_+)$.

2. Find the first leaf (for R'_1) and the last leaf (for R'_m) of $S(L_+)$, in $O(\log m)$ time. If R'_1 and R'_m have different colors, then proceed to the next step. Otherwise R'_1 and R'_m have the same color (say, it is red). For each integer s, $1 \leq s \leq m$, let $f(s)$ be the number of red elements minus the number of blue elements in the set $\{R'_1, R'_2, \ldots, R'_s\}$; observe that $|f(s+1) - f(s)| = 1$ and that in this case $f(1) = 1$ whereas $f(m-1) = -1$. This implies, by a simple "continuity" argument, that there is some integer ℓ, $1 < \ell < m-1$, for which $f(\ell) = 0$. (A somewhat similar continuity argument was used in [3] in the context of matching points.) Next, we shall search for such an ℓ in time $O(\min\{\ell, m-\ell\})$ rather than in time $O(m)$, as follows. We linearly search for it along the leaf sequence of $S(L_+)$, by two interleaved searches: One starting from the beginning of L_+, from R'_1 up, and the other starting from the end of L_+, from R'_{m-1} down, where we alternate between the two searches until one of them first hits a desired value ℓ which we know must exist. Hence, we find an ℓ value for which $f(\ell) = 0$ in $O(\min\{\ell, m-\ell\})$ time, rather than in $O(m)$ time. This defines two subproblems L' and L'': $L' = \{R'_1, R'_2, \ldots, R'_\ell\}$ and $L'' = \{R'_{\ell+1}, R'_{\ell+2}, \ldots, R'_m\}$. In $O(\log m)$ time, we split $S(L_+)$ into $S(L')$ and $S(L'')$. Then we recursively call **MATCH**(L') and **MATCH**(L'').

   Analysis: This step has a cumulative total cost of $O(n \log n)$ time rather than $O(n^2)$ even though the two subproblems so generated and solved recursively can be very "unbalanced", e.g., $|L'|$ could be $O(1)$. The analysis is as follows: We spend only $O(\log m + \min\{\ell, m-\ell\})$ time in generating the two subproblems; we can "charge" the $\log m$ term of this cost to the recursive call itself (i.e., to the node of that recursive call in the recursion tree), and the $\min\{\ell, m-\ell\}$ term to the rectangles of the smaller subproblem ($O(1)$ time per rectangle). A rectangle that is so "charged" ends up in a subproblem of no more than half the size of its previous subproblem, and hence cannot be charged more than $\log n$ times, for a total (over all the $2n$ rectangles of R) of $O(n \log n)$. The total number of nodes in the recursion tree is $O(n)$, and hence the overall cost of the charges to the nodes of that recursion tree ($\log m$ per node) is $O(n \log n)$.

3. R'_1 and R'_m have different colors. Obtain, from the label at the root of $S(L_+)$, the smallest rectangle of L_+ according to the L_- ordering. Let R'' be this rectangle. Rectangle R'' must have the same color as one of $\{R'_1, R'_m\}$, so suppose WLOG that it has the same color as R'_1. Then we (i) match R'_1 and R'', (ii) delete R'_1 and R'' from $S(L_+)$ in $O(\log m)$ time, and (iii) recursively solve the problem on the resulting L_+.

Comment: The path that will join the pair just matched will be along the boundary of $CH(L_+ - \{R_1', R''\})$. The justification for the monotonicity of this path follows from Lemmas 3 and 4, which ensure that the path from R_1' to R'' along the boundary of $CH(L_+ - \{R_1', R''\})$ consists of at most two subpaths: An increasing staircase followed by a decreasing staircase. This step also has a cumulative total cost of $O(n \log n)$ time, because each of the n matched pairs is charged a cost of $O(\log n)$ time by the step.

As analyzed above, algorithm **MATCH** computes n matched pairs of red/blue rectangles of R in $O(n \log n)$ time and $O(n)$ space.

6 Reporting the Actual Paths

This section shows how to output the actual monotone paths between all the n matched red/blue rectangle pairs in $O(n \log n + \lambda)$ time, where λ is the total number of segments that make up these n paths.

Recall the comments we made after a rectangle pair was matched by the algorithm of the previous section (specifically, following Steps 1 and 3). These comments described the desired path between the pair just matched in terms of a rectilinear convex hull $CH(v)$ of a subproblem associated with a particular place (i.e., a node) v in the recursion tree of algorithm **MATCH** at which this subproblem occurred. We postponed the actual computation of these $CH(v)$ convex hulls, because once we have the overall structure of the recursion tree, we can traverse it and compute these $CH(v)$ hulls bottom up, with insertion operations only (since the subproblem of a child node in the recursion tree is that of its parent node *minus* some rectangles). Thus, this enables us to use the fact that maintaining rectilinear convex hulls, in the face of insertions only, is possible in logarithmic time per insertion [31].

Hence, the idea is to run the matching algorithm of Section 5 and make sure that, after that algorithm has executed, it leaves behind the skeleton of its recursion tree, which we call *RecTree*, together with certain information describing how a path between a matched rectangle pair is related to $CH(v)$ (i.e., the description in the "comments" of algorithm **MATCH**). This description information uses $O(1)$ space per matched pair. This skeleton just gives the overall structure of *RecTree.* It does not store directly the rectangles of the subproblem associated with each node v of *RecTree* (that would be too expensive in terms of the space complexity), but rather how the rectangles of v are related to those of v's children:

1. If v has only one child in *RecTree*, then its associated rectangles are those of its only child plus two rectangles that are matched by algorithm **MATCH** at v: It is these two rectangles that are explicitly stored at v in *RecTree*.

2. If v has two children in *RecTree*, then its associated rectangles are the union of the rectangles of both its children.

In either case, we store $O(1)$ information at each node v, so that *RecTree* uses altogether $O(n)$ space. The problem of computing the actual monotone path (if any) associated with each node v in *RecTree* clearly reduces to computing $CH(v)$ in turn and using it to print that path. The computation of the $CH(v)$'s associated with all the nodes v of *RecTree* is done by a simple traversal of *RecTree* during which the $CH(v)$'s are computed according to the postorder numbers [1] of the nodes v in *RecTree*. Of course, at a node v of *RecTree* that has two children (say, u and w), we do not create $CH(v)$ by individually inserting the vertices of $CH(u)$ into $CH(w)$, but rather we obtain $CH(v)$ by "merging" $CH(u)$ and $CH(w)$ in logarithmic time [31]. After $CH(v)$ is computed, the actual path between the matched rectangle pair of node v is computed by walking along $CH(v)$, in time proportional to the size of the path plus a logarithmic additive term. We assume that if two such matching paths share some common portions on certain convex hulls so computed, then the two paths are apart by at least a positive distance that can be made arbitrarily small. The overall time of this algorithm is therefore $O(n \log n)$ plus the time needed to print all the output paths, i.e., $O(\lambda)$.

7 Extensions to Monotone Polygonal Obstacles

In this section, we extend our techniques for matching red/blue isothetic rectangle obstacles to matching red/blue polygonal obstacles in the plane that are all monotone with respect to a coordinate axis (say, the y-axis). Let W be a set of r red and r blue disjoint polygonal obstacles in the plane, with a total of n vertices. We assume that all the polygonal obstacles in W are monotone to the y-axis, and call them *y-monotone polygons*. We show that it is possible to match all the red and blue polygons in W, by connecting the r matched red/blue polygon pairs with r mutually disjoint paths. The properties of the matching paths are similar to those for isothetic rectangles, except for the monotonicity: In this case, a path can be used for the matching if it can be partitioned into at most *two* subpaths, each of which is monotone to the y-axis. Our algorithms for computing such a matching have the same complexity bounds as the matching algorithms for isothetic rectangles in the previous sections.

One consequence of considering y-monotone polygonal obstacles (whose structures are less nice than those of isothetic rectangles) is that we must use a weaker monotonicity constraint on the matching paths. This is because even with a geometric setting consisting of disjoint *convex* polygonal obstacles in the plane, there is in general no obstacle-avoiding path between two arbitrary points that is monotone to the x-axis or to the y-axis. But in such a setting, a

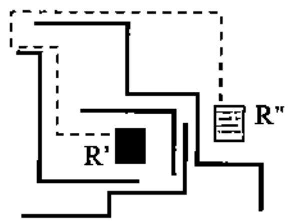

Figure 8: A path with two y-monotone subpaths among rectilinear convex obstacles.

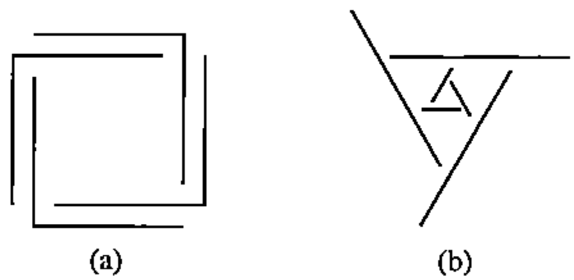

Figure 9: There is no staircase separator for rectilinear and non-rectilinear convex obstacles.

path consisting of at most two y-monotone subpaths always exists between any two points (see Figure 8 for an example). Another consequence of considering y-monotone polygonal obstacles is that there is in general no staircase separator for partitioning such geometric object sets. In the two examples of Figure 9, there exists no staircase (even with respect to *any* two orthogonal lines) that partitions each *convex* obstacle set into two subsets, such that every subset contains more than one obstacle. However, as we will show, there exist y-monotone paths that partition y-monotone polygons. Note that a key difference between staircases and y-monotone paths is that staircases are monotone to *both* the x-axis and y-axis, while y-monotone paths need not be monotone to the x-axis.

It turns out that the matching algorithms based on the geometric structures of y-monotone polygonal obstacles are similar to and in fact simpler than the matching algorithms for isothetic rectangles. Also, although we have chosen in this section to focus our discussion on rectilinear geometric objects (obstacles, paths, etc), it is actually not difficult to modify our algorithms so that they will work with non-rectilinear objects under the y-monotonicity constraint.

Let the obstacle set $W = \{W_0, W_1, \ldots, W_{2r}\}$, where W_0 is the extra "dummy" rectangle R_0 to the left of all the other obstacles in W (as introduced in Section 3). We first preprocess W as in Section 3. From the left vertex of the lowest edge of every W_i, shoot a leftwards-moving horizontal ray r_i; let *Parent*(i) be W_j, where W_j is the first obstacle in W hit by the ray r_i. Maintain for every W_j an adjacency list of all the obstacles in W whose *Parent* is W_j, sorted by the *decreasing* y-coordinates of their leftwards-moving horizontal rays. This gives a tree structure whose nodes are the obstacles in W (as the tree T in Section 3) and which we again denote

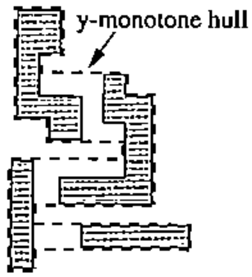

Figure 10: An example of the y-monotone hull of a set of obstacles.

by T. Label the nodes of T by their preorder numbers in T, and re-label the obstacles in W by their corresponding preorder numbers in T. This preprocessing can be done by a horizontal trapezoidal decomposition of W [33] and a preorder traversal of T [1], in altogether $O(n \log n)$ time and $O(n)$ space. WLOG, let i be the label of W_i in the preprocessed form. In addition, we also construct, as part of the preprocessing, the planar subdivision [33] that is defined by the horizontal trapezoidal decomposition of W. The construction of this planar subdivision also takes $O(n \log n)$ time and $O(n)$ space.

For any consecutive subset $W' = \{W_i, W_{i+1}, \ldots, W_j\}$ of W, where $i > 0$, we define the *y-monotone hull* of W', denoted by $CH_y(W')$, to be the region with the smallest area that contains all the obstacles in W' and that is y-monotone (see Figure 10 for an example). Note that the region $CH_y(W')$ so defined may be disconnected. If this is the case, we assume that we link the connected components of $CH_y(W')$ together with some paths of zero width, so that $CH_y(W')$ becomes connected and is still y-monotone.

Note that the boundary of every y-monotone polygon can be easily partitioned into two y-monotone paths, which we call the *left boundary* and *right boundary* of such a polygon. For every point p in the plane that is to the right of the root obstacle W_0 of T and is not in the interior of any obstacle, we define the path $Q(p)$ from p to W_0 as in Section 3, with one small exception: When $Q(p)$ follows a leftwards-moving horizontal ray and hits an obstacle $W_i \neq W_0$, $Q(p)$ goes to the left vertex of the lowest edge of W_i along a downwards y-monotone path on the right boundary of W_i. $Q(p)$ so defined is clearly a unique y-monotone path, although it need not be x-monotone simultaneously.

The following observations are analogous to those of Lemmas 1 and 2 and Theorem 1. The differences in these observations and their proof arguments stem from the structural differences between the convex hulls of isothetic rectangles and the y-monotone hulls of y-monotone polygons in our matching problems.

Lemma 5 *For an obstacle W_i in $W - \{W_0\}$, let p and q be two points such that p is on the left boundary of W_i and q is on the right boundary of W_i. Then no point of $Q(p)$ is strictly below* *any point of $Q(q)$.*

Proof. A crucial fact to the proof is that both $Q(p)$ and $Q(q)$ are planar y-monotone paths. The proof argument is similar to that of Lemma 1. $\square$

Lemma 6 *Let p and q be two points in the plane such that p is on the left boundary of an obstacle W_i and q is on the right boundary of W_i, with $i > 0$. Let u (resp., v) be the left vertex of the lowest edge of an obstacle W_a (resp., W_b), such that u (resp., v) is on $Q(p)$ (resp., $Q(q)$) but not on $Q(q)$ (resp., $Q(p)$). Then the preorder number of W_a in the tree T of obstacles is smaller than that of W_b, i.e., $a < b$.*

Proof. This follows from Lemma 5 and from the definition of the tree T. $\square$

Theorem 2 *Given a preprocessed set W of $2r$ disjoint y-monotone polygonal obstacles with n vertices in total, the subsets $\{W_1, W_2, \ldots, W_k\}$ and $\{W_{k+1}, W_{k+2}, \ldots, W_{2r}\}$, for any integer k with $1 \leq k < 2r$, form a partition of the set W that has the desired property, that is, there exists an obstacle-avoiding y-monotone path of size $O(n)$ that separates these two subsets. Furthermore, such a y-monotone path can be computed in $O(n)$ time.*

Proof. Let $W(a,b)$ denote the subset $\{W_a, W_{a+1}, \ldots, W_b\}$ of W. For the existence of such a y-monotone path, we first show that for any $i < j$, the following holds: (1) $CH_y(W(1,i))$ does not intersect W_j, and (2) $CH_y(W(j,2r))$ does not intersect W_i. We give the proof only for (1), that for (2) being similar.

We prove (1) by contradiction: Suppose to the contrary that for some $j > i$, W_j intersects $CH_y(W(1,i))$. Then for a point $w \in CH_y(W(1,i)) \cap W_j$, there must be a point z of a W_s, $s \leq i < j$, such that $y(w) = y(z)$ and $x(w) < x(z)$, (i.e., z is strictly to the right of w). (If such a point z did not exist, then w would have not belonged to $CH_y(W(1,i))$ by the definition of y-monotone hulls, a contradiction.) WLOG, let $z \in W_s$ be the leftmost such point. Then z must be on the left boundary of W_s and the leftwards-moving horizontal ray from the left vertex of the lowest edge of W_s cannot first hit W_j (otherwise, we would have a contradiction). Let z' be a point on the right boundary of W_s such that $y(z) > y(z')$. Then by Lemma 6, the preorder number of W_j in T is smaller than that of W_s, a contradiction.

We can compute a desired y-monotone path by letting the path first go along the right boundary of $CH_y(W(1,k))$ as much as possible, then along the left boundary of $CH_y(W(k+1,2r))$ (if necessary), and finally extend vertically upwards and downwards to infinity. The y-monotone path so obtained clearly has a size of $O(n)$. Given the planar subdivision based on the horizontal trapezoidal decomposition of the obstacle set W (this subdivision is part of the preprocessing result), it is possible to obtain such a y-monotone path in $O(n)$ time. This is done by examining the $O(n)$ cells of the planar subdivision to identify those cells that separate the two subsets $W(1,k)$ and $W(k+1,2r)$, i.e., the cells whose left (resp., right) boundaries are on the right (resp., left) boundaries of the polygons in $W(1,k)$ (resp., $W(k+1,2r)$). □

Note that in a fashion similar to Theorem 2, we can also partition the preprocessed set W into two subsets based on the total sizes of the polygons in the resulting subsets. That is, for an integer j with $1 \leq j < n$, we can partition the preprocessed obstacle set W into two subsets $W(1,k)$ and $W(k+1,2r)$ with a y-monotone path, such that the total number of polygon vertices of $W(1,k)$ is no bigger than j but the total number of polygon vertices of $W(1,k+1)$ is strictly larger than j. This partitioning can also be done in $O(n)$ time.

Theorem 2 enables us to obtain efficient algorithms for computing a desired matching for y-monotone polygons, as did Theorem 1 for isothetic rectangles. In fact, the matching algorithms for y-monotone polygons are similar to and actually simpler than the ones for isothetic rectangles.

Like the matching algorithms for isothetic rectangles, the algorithms here also maintain the list L_+. However, unlike the algorithms for isothetic rectangles, L_+ here is always a consecutive sublist of the original list $W(1,2r)$ and is maintained only as a doubly linked list. Further, the algorithms here do not need to use the tree T' and hence the list L_-, and do not use the 2-3 tree $S(L_+)$. We only sketch below the computation of these algorithms, since they are very similar to those of Sections 5 and 6.

To specify the matching pairs of the red/blue polygons in a list $L_+ = (W'_1, W'_2, \ldots, W'_m)$ (without computing the actual paths), the algorithm simply does the following:

> If W'_1 and W'_m are of different colors, then match W'_1 and W'_m (by letting the W'_1-to-W'_m path go along first the left boundary of $CH_y(L_+)$ and then the right boundary of $CH_y(L_+)$), and recursively solve the problem on $L_+ - \{W'_1, W'_m\}$ if $L_+ - \{W'_1, W'_m\}$ is non-empty; otherwise, partition L_+ into two consecutive sublists (as in Step 2 of algorithm **MATCH**) and recursively solve the two subproblems.

A matching path so specified consists of at most two y-monotone subpaths because it follows first the left boundary and then the right boundary of a y-monotone hull. As analyzed in Section 5 for algorithm **MATCH**, the matching algorithm here takes $O(r \log r)$ time after the ordered list $W(1,2r)$ is made available by the $O(n \log n)$ time preprocessing.

The algorithm for computing the r actual paths of a matching here is similar to the one for isothetic rectangles in Section 6: It maintains the recursion tree *RecTree* of the above matching algorithm, and computes the y-monotone hull $CH_y(v)$ for the subproblem on every node v of *RecTree*. Each of the left and right boundaries of $CH_y(v)$ can be maintained by a

2-3 tree. The geometric structures of the y-monotone hulls of the input polygons in $RecTree$ can be exploited by our computation in the following way: When we need to "merge" two y-monotone hulls $CH_y(u)$ and $CH_y(w)$ to obtain $CH_y(v)$ (with u and w being the left and right children of v, respectively), we replace the corresponding portions of the (say) left boundary of $CH_y(w)$ by the left boundary of each connected component of $CH_y(u)$ (if $CH_y(u)$ indeed consists of more than one connected component). This can be done by using $O(1)$ split and concatenation operations of 2-3 trees for each component of $CH_y(u)$, in logarithmic time. Since we can charge the time for "merging" each such connected component to a horizontal line segment of the horizontal trapezoidal decomposition and since there are $O(n)$ such line segments in the trapezoidal decomposition, the total time for our algorithm to output all the r actual paths between the matched red/blue polygon pairs is $O(n \log n + \lambda)$, where λ is the total number of segments that make up these r paths. The space bounds of the matching algorithms in this section are $O(n)$.

8 Lower Bounds for the Matching Problems

In this section, we prove $\Omega(n \log n)$ lower bounds in the algebraic computation tree model [8] for the matching problems studied in this paper.

First, we show that the problem of matching $2n$ disjoint red/blue isothetic rectangles with nonintersecting monotone rectilinear paths in the plane requires $\Omega(n \log n)$ time in the worst case. Actually, we will show an $\Omega(n \log n)$ lower bound for the following (simpler) problem **P**: Giving n red and n blue disjoint isothetic rectangles in the plane, find a monotone rectilinear obstacle-avoiding path from a *specified* red rectangle (say, R_1) to some (unspecified) blue rectangle V_i. The reason for considering problem **P** is that this problem can be easily reduced to our matching problem since any solution to the matching problem definitely contains such a monotone path between the red rectangle R_1 and some blue rectangle V_i. The key to our proof is a reduction from the problem of sorting $O(n)$ pairwise distinct positive integers (in an arbitrary range) to problem **P**. Note that based on Yao's $\Omega(n \log n)$ lower bound result for the element uniqueness problem on n arbitrary integers [42], Chen, Das, and Smid [10] showed that sorting $O(n)$ pairwise distinct positive integers in the worst case requires $\Omega(n \log n)$ time in the algebraic computation tree model.

The reduction goes as follows. Consider a set K of n pairwise distinct positive integers I_1, I_2, ..., I_n. Let I_a (resp., I_b) be the smallest (resp., largest) integer in the set K (it is easy to find I_a and I_b in $O(n)$ time). WLOG, assume that $I_a > 2$. For every integer $I_j \in K$, map I_j to a set U_j of four *red* isothetic rectangles R_l^j, R_r^j, R_u^j, and R_d^j in the plane, as follows (see Figure

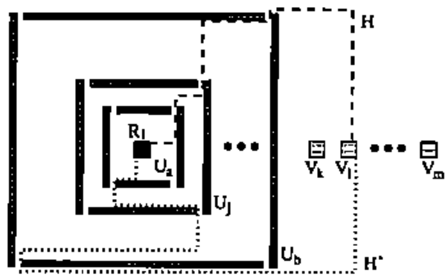

Figure 11: Illustrating the reduction of the lower bound proofs.

11): The shorter edges of all the four red rectangles in U_j have the same length of 0.5 units; the right (resp., left) edge of R_r^j (resp., R_l^j) has the point $(I_j, 0)$ (resp., $(-I_j, 0)$) as its middle point and has a length of $2I_j$, while the top (resp., bottom) edge of R_u^j (resp., R_d^j) has the point $(0, I_j)$ (resp., $(0, -I_j)$) as its middle point and has a length of $2I_j - 1 - 2\epsilon$, for a very small fixed $\epsilon > 0$. Let R_1 be a red isothetic unit box whose center is at the origin of the coordinate system. We then have $4n + 1$ red rectangles. We next create $4n + 1$ isothetic blue rectangles V_l's in the following way: These blue rectangles are all isothetic unit boxes whose centers are all on the x-axis; every two consecutive blue boxes are one unit distance apart, and the leftmost blue box is at least one unit distance to the right of U_b (see Figure 11). It is clear that the $O(n)$ red/blue isothetic rectangles so obtained are pairwise disjoint (since the input integers are pairwise distinct), and that the construction of this rectangle set takes $O(n)$ time.

Now, it is an easy matter to observe that (1) an R_1-to-V_i path in this setting can be monotone only to the x-axis (but not to the y-axis), and (2) *any* such monotone R_1-to-V_i path must get around every red rectangle set U_j in the sorted order of the corresponding I_j values of the U_j's (Figure 11). Let H be a monotone rectilinear R_1-to-V_i path computed by any algorithm for problem **P**, with $|H| = O(n)$. We assume that when the path H is getting around a particular rectangle set U_j, it picks up the index j and associates j with the horizontal edge of H that contains the x-coordinate of the rightmost edge of U_j. Then given such a path H, we can output the sorted sequence of the input integers in K by tracing H and picking up the indices of the integers I_j from their associated horizontal edges of H along the path order of H. Such a tracing of H can be easily done in $O(n)$ time. This completes the lower bound proof for problem **P**.

Our lower bound proof for the matching problem on y-monotone polygons uses the same reduction construction as for that on isothetic rectangles, except that we now compute a path which consists of at most *two* y-monotone subpaths instead of one monotone path. That is, we use any algorithm for computing such an R_1-to-V_i path among y-monotone polygons to build a geometric sorting device for integer input; the reduction is the same as the one illustrated in Figure 11 and takes $O(n)$ time. This reduction works because *any* R_1-to-V_i obstacle-avoiding rectilinear path H' that consists of at most two y-monotone subpaths in the setting of Figure 11 must get around every red rectangle set U_j in the sorted order of the corresponding I_j values of the U_j's. Therefore, such a path H' can be used to report the sorted sequence of the input integers in $O(n)$ time, implying an $\Omega(n \log n)$ lower bound for the matching problem on y-monotone polygons.

9 Further Remarks

As mentioned earlier, Theorem 1 implies an efficient parallel bound for equipartitioning a set of disjoint isothetic rectangles. This fact is potentially useful in the parallel algorithmics of other, not necessarily red/blue, rectangle problems (as is clear from [4, 5], where tremendous simplifications follow from the next theorem). Therefore, this useful side-effect of Theorem 1 is summarized below.

Theorem 3 *Let R be a set of $2m$ disjoint isothetic rectangles (not given in any particular order). Then an m-processor CREW PRAM can compute, in $O(\log m)$ time, an increasing staircase S that does not intersect the interior of any rectangle in R and partitions R into two equal parts, with $|S| = O(m)$.*

Proof. This follows from Theorem 1 and the fact that a trapezoidal decomposition [6] as well as the preorder numbers in a tree [37] can all be computed in parallel within these bounds. □

In fact, the preprocessed form of R required by Theorem 1 can be obtained as a by-product of Theorem 3, in $O(\log m)$ time using m CREW PRAM processors. Once this form is available, we can do a little more than Theorem 3: We can partition the set $R = \{R_1, R_2, \ldots, R_{2m}\}$ into two subsets $\{R_1, R_2, \ldots, R_k\}$ and $\{R_{k+1}, R_{k+2}, \ldots, R_{2m}\}$, for any integer k with $1 \leq k < 2m$, in $O(\log t)$ time using $t/\log t$ processors in the CREW PRAM or even the EREW PRAM model [22], where $t = \min\{k, 2m-k\}$. This is done by using, instead of the two-way divide-and-conquer algorithm given in the proof of Theorem 1, a many-way divide-and-conquer approach as in [9, 20]. The details of this parallel algorithm are very similar to (and in fact even simpler than) those of [9, 20], and hence are omitted.

The following partition result may also be useful to designing parallel algorithms for certain geometric problems.

Theorem 4 *Let W be a set of $2r$ disjoint y-monotone polygons (not given in any particular order) with a total of m vertices. Then an m-processor CREW PRAM can compute, in $O(\log m)$ time, a y-monotone path P that does not intersect the interior of any polygon in W and partitions W into two subsets of r polygons each, with $|P| = O(m)$.*

Proof. This follows from Theorem 2 and the fact that a trapezoidal decomposition and the planar subdivision based on it [6] as well as the preorder numbers in a tree [37] can all be computed in parallel within these bounds. □

Again, we can also preprocess W in $O(\log m)$ time using m CREW PRAM processors. After that, such a y-monotone path P, as defined in Theorem 4, can be obtained in $O(\log m)$ time using $m/\log m$ CREW PRAM processors. This is done by first examining the cells of the planar subdivision (to identify those cells that separate the two subsets of the polygons in W) and then using parallel list ranking [22] to find the path P. Note that it is also possible to modify Theorem 4 to partition W into two subsets based on the total sizes of the polygons in the resulting subsets.

We conclude with an implementation note about our algorithms. If we are to program the matching algorithms for isothetic rectangles, we would modify them by creating (in Step 2) $S(L')$ and $S(L'')$ only as a last resort, by inserting before Step 2 a Step 1' in which we check whether R'_1 and R'_2 are of different colors — if so we match them, delete them, etc, and if not we check whether R'_{m-1} and R'_m are of different colors — if so we match them, delete them, etc, and if not we go to Step 2. Thus, we go to Step 2 only if we are unable to match the pair $\{R'_1, R'_2\}$ and the pair $\{R'_{m-1}, R'_m\}$. Performing such a Step 1' before Step 2 gives preference to short paths over long ones, since an R'_1-to-R'_m path is likely to be longer than an R'_1-to-R'_2 (or R'_{m-1}-to-R'_m) path. For y-monotone polygons, an efficient heuristic that may produce short paths for a matching we desire is to use a modification of the so called red/blue matching approach [2, 23] for matching red/blue elements in an ordered list (in our situation, the ordered list is $W(1, 2r)$). Of course, this assumes that short paths are practically better than long ones.

The above discussion suggests the obvious open problems of finding matchings that satisfy some additional length criteria, such as:

- Minimum sum of lengths of all n paths, or
- Minimum maximum length of all n paths, or
- Versions of the above two where "length" means number of links rather than the usual L_1 length (hence this version of the sum-of-lengths problem amounts to minimizing what we earlier called λ).

References

[1] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, Mass., 1974.

[2] M. G. Andrews, M. J. Atallah, D. Z. Chen, and D. T. Lee, "Parallel algorithms for maximum matching in interval graphs," *Proc. 9th IEEE International Parallel Processing Symp.*, 1995, pp. 84–92.

[3] M. J. Atallah, "A matching problem in the plane," *J. of Computer and Systems Sciences*, 31 (1985), pp. 63–70.

[4] M. J. Atallah and D. Z. Chen, "Parallel rectilinear shortest paths with rectangular obstacles," *Computational Geometry: Theory and Applications*, 1 (1991), pp. 79–113.

[5] M. J. Atallah and D. Z. Chen, "On parallel rectilinear obstacle-avoiding paths," *Computational Geometry: Theory and Applications*, 3 (1993), pp. 307–313.

[6] M. J. Atallah, R. Cole, and M. T. Goodrich, "Cascading divide-and-conquer: A technique for designing parallel algorithms," *SIAM J. Computing*, 18 (1989), pp. 499–532.

[7] M. J. Atallah, S. E. Hambrusch, and L. E. TeWinkel, "Parallel topological sorting of features in a binary image," *Algorithmica*, 6 (1991), pp. 762–769.

[8] M. Ben-Or, "Lower bounds for algebraic computation trees," *Proc. 15th Annual ACM Symp. on Theory of Computing*, 1983, pp. 80–86.

[9] D. Z. Chen, "Efficient geometric algorithms on the EREW PRAM," *IEEE Trans. on Parallel and Distributed Systems*, 6 (1) (1995), pp. 41–47.

[10] D. Z. Chen, G. Das, and M. Smid, "Lower bounds for computing geometric spanners and approximate shortest paths," *Proc. 8th Canadian Conf. on Computational Geometry*, 1996, pp. 155–160.

[11] D. Z. Chen and K. S. Klenk, "Rectilinear short path queries among rectangular obstacles," *Information Processing Letters*, 57 (6) (1996), pp. 313–319.

[12] D. Z. Chen, K. S. Klenk, and H.-Y. T. Tu, "Shortest path queries among weighted obstacles in the rectilinear plane," *Proc. 11th Annual ACM Symp. Computational Geometry*, 1995, pp. 370–379.

[13] J. Choi and C.-K. Yap, "Rectilinear geodesics in 3-space," *Proc. 11th Annual ACM Symp. Computational Geometry*, 1995, pp. 380–389.

[14] K. L. Clarkson, S. Kapoor, and P. M. Vaidya, "Rectilinear shortest paths through polygonal obstacles in $O(n(\log n)^2)$ time," *Proc. 3rd Annual ACM Symp. Computational Geometry*, 1987, pp. 251–257.

[15] K. L. Clarkson, S. Kapoor, and P. M. Vaidya, "Rectilinear shortest paths through polygonal obstacles in $O(n \log^{3/2} n)$ time," manuscript.

[16] W. Dai, T. Asano, and E. S. Kuh, "Routing region definition and ordering scheme for building-block layout," *IEEE Trans. on Computer-Aided Design*, CAD-4 (3) (1985), pp. 189–197.

[17] M. de Berg, M. van Kreveld, and B. J. Nilsson, "Shortest path queries in rectangular worlds of higher dimension," *Proc. 7th Annual Symp. Computational Geometry*, 1991, pp. 51–59.

[18] P. J. de Rezende, D. T. Lee, and Y. F. Wu, "Rectilinear shortest paths in the presence of rectangles barriers," *Discrete & Computational Geometry*, 4 (1989), pp. 41–53.

[19] H. ElGindy and P. Mitra, "Orthogonal shortest route queries among axes parallel rectangular obstacles," *International J. of Computational Geometry and Applications*, 4 (1) (1994), pp. 3–24.

[20] M. T. Goodrich, "Finding the convex hull of a sorted point set in parallel," *Information Processing Letters*, 26 (1987/88), pp. 173–179.

[21] M. Iwai, H. Suzuki, and T. Nishizeki, "Shortest path algorithm in the plane with rectilinear polygonal obstacles" (in Japanese), *Proc. of SIGAL Workshop*, July 1994.

[22] J. JáJá, *An Introduction to Parallel Algorithms*, Addison-Wesley, Reading, MA, 1992.

[23] S. K. Kim, "Optimal parallel algorithms on sorted intervals," *Proc. 27th Annual Allerton Conf. Communication, Control, and Computing*, 1989, pp. 766–775.

[24] R. C. Larson and V. O. Li, "Finding minimum rectilinear distance paths in the presence of barriers," *Networks*, 11 (1981), pp. 285–304.

[25] D. T. Lee, T. H. Chen, and C. D. Yang, "Shortest rectilinear paths among weighted obstacles," *International J. of Computational Geometry and Applications*, 1 (2) (1991), pp. 109–124.

[26] D. T. Lee, C. F. Shen, C. D. Yang, and C. K. Wong, "Non-crossing paths problems," manuscript, Dept. of EECS, Northwestern University, 1991.

[27] J. S. B. Mitchell, "An optimal algorithm for shortest rectilinear path among obstacles," *First Canadian Conf. on Computational Geometry*, 1989.

[28] J. S. B. Mitchell, "L_1 shortest paths among polygonal obstacles in the plane," *Algorithmica*, 8 (1992), pp. 55–88.

[29] P. Mitra and B. Bhattacharya, "Efficient approximation shortest-path queries among isothetic rectangular obstacles," *Proc. 3rd Workshop on Algorithms and Data Structures*, 1993, pp. 518–529.

[30] T. M. Nicholl, D. T. Lee, Y. Z. Liao, and C. K. Wong, "On the X-Y convex hull of a set of X-Y polygons," *BIT*, 23 (4) (1983), pp. 456–471.

[31] M. H. Overmars and J. van Leeuwen, "Maintenance of configurations in the plane," *J. of Computer and Systems Sciences*, 23 (1981), pp. 166–204.

[32] E. Papadopoulou, "k-Pairs non-crossing shortest paths in a simple polygon," to appear in the *7th Annual International Symp. on Algorithms and Computation*, 1996, Osaka, Japan.

[33] F. P. Preparata and M. I. Shamos, *Computational Geometry: An Introduction*, Springer-Verlag, New York, 1985.

[34] F. P. Preparata and K. J. Supowit, "Testing a simple polygon for monotonicity," *Information Processing Letters*, 12 (1981), pp. 161–164.

[35] J. Takahashi, H. Suzuki, and T. Nishizeki, "Algorithms for finding non-crossing paths with minimum total length in plane graphs," *Proc. 3rd Annual International Symp. on Algorithms and Computation*, 1992, pp. 400–409.

[36] J. Takahashi, H. Suzuki, and T. Nishizeki, "Finding shortest non-crossing rectilinear paths in plane regions," *Proc. 4th Annual International Symp. on Algorithms and Computation*, 1993, pp. 98–107.

[37] R. E. Tarjan and U. Vishkin, "Finding biconnected components and computing tree functions in logarithmic parallel time," *SIAM J. Computing*, 14 (1985), pp. 862–874.

[38] P. Widmayer, Y. F. Wu, and C. K. Wong, "On some distance problems in fixed orientations," *SIAM J. Computing*, 16 (4) (1987), pp. 728–746.

[39] Y. F. Wu, P. Widmayer, M. D. F. Schlag, and C. K. Wong, "Rectilinear shortest paths and minimum spanning trees in the presence of rectilinear obstacles," *IEEE Trans. on Computers*, C-36 (1987), pp. 321–331.

[40] C. D. Yang, T. H. Chen, and D. T. Lee, "Shortest rectilinear paths among weighted rectangles," *Journal of Information Processing*, 13 (4) (1990), pp. 456–462.

[41] C. D. Yang, D. T. Lee, and C. K. Wong, "Rectilinear path problems among rectilinear obstacles revisited," *SIAM J. Computing*, 24 (3) (1995), pp. 457–472.

[42] A. C.-C. Yao, "Lower bounds for algebraic computation trees with integer inputs," *SIAM J. Computing*, 20 (1991), pp. 655–668.