Minimum Rectangular Partition Problem for Simple Rectilinear Polygons

W. T. LIOU, JIMMY JIANN-MEAN TAN, AND R. C. T. LEE, FELLOW, IEEE

Abstract—**In this paper, an $O(n \log \log n)$ algorithm is proposed for minimally rectangular partitioning a simple rectilinear polygon. For any simple rectilinear polygon P, a vertex-edge visible pair is a vertex and an edge that can be connected by a horizontal or vertical line segment that lies entirely inside P. We show that, if the vertex-edge visible pairs are found, the maximum matching and the maximum independent set of the bipartite graph derived from the chords of a simple rectilinear polygon can be found in linear time without constructing the bipartite graph. Using our algorithm, the minimum partition problem for convex rectilinear polygons and vertically (horizontally) convex rectilinear polygons can be solved in $O(n)$ time.**

I. Introduction

THE minimum rectangular partition problem for a simple rectilinear polygon is to partition the interior of a simple rectilinear polygon into a minimum number of rectangles. The decomposition can be classified into two types, depending on the resulting rectangles. If the resulting rectangles cannot overlap with each other, then the decomposition is a *partition*. If the resulting rectangles overlap with each other, then the decomposition is a *cover*. Both partitioning approach and covering approach have been discussed in previous researches such as [3], [7], [10], [11], [14], [15] for partitioning problems and [2], [4], [8] for covering problems. In this paper, we shall consider the minimum partition problem for simple rectilinear polygons. We shall use both horizontal and vertical cuts to find a minimum partition. However, in some applications, only horizontal cuts are permissible [14].

This problem has been studied in [3], [10], [11], [15], [16]. The best-known result is proposed by [10], which requires $O(n^{1.5}\log n)$ time. Note that the algorithms of previous approaches can be applied to a rectilinear polygon with holes. In this paper, we propose an algorithm for partitioning simple rectilinear polygons, which requires $O(n \log \log n)$ time. We do not know yet whether or not our approach can be applied to a rectilinear polygon with holes.

Manuscript received August 8, 1988; revised April 14, 1989. This work was supported in part by the National Science Council of the Republic of China under Grant NSC-77-0201-E007-04. This paper was recommended by Associate Editor R. H. J. M. Otten.

W. T. Liou is with the Department of Management Information Systems, National Cheng Chi University, Taipei, Taiwan.

J. J.-M. Tan is with the Institute of Information Science, National Chiao Tung University, Hsinchu, Taiwan.

R. C. T. Lee is with the National Tsing Hua University, Hsinchu, Taiwan, and the Academia Sinica, Taipei, Taiwan.

IEEE Log Number 9035475.

Based on our approach for solving the partition problem for a simple rectilinear polygon, the partition problem for convex rectilinear polygons or vertically (horizontally) convex polygons can be solved in linear time, which is optimal.

The rest of this paper is organized as follows. In Section II, we introduce some results of previous research. In Section III, we introduce a theorem of Lipski and Preparata [12] and prove that this theorem can be applied to the bipartite graph derived from chords of a simple rectilinear polygon. In Section IV, we propose an algorithm to find the maximum matching of vertical and horizontal chords for horizontally convex rectilinear polygons without actually constructing any bipartite graph. In Section V, the algorithm in Section IV is extended to simple rectilinear polygons. In Section VI, based on the maximum matching found in Section V, we find the maximum nonintersecting chords which can be used to determine a minimal partition. The total time required for executing the algorithms in Sections V and VI is $O(n \log \log n)$. Section VII gives the conclusion.

II. Previous Results

The minimum rectangular partition problem has been studied in [3], [10], [11], [15], [16]. Some of their results are discussed; these are the starting point of our research.

A *rectilinear polygon* on the plane is a polygon whose sides are either vertical or horizontal. A *simple rectilinear polygon* is a rectilinear polygon which has no windows (holes) in it. The *minimum rectangular partition problem* defined on a simple rectilinear polygon can be stated as follows: given a simple rectilinear polygon P on the plane, find a minimally sized set of nonoverlapping rectangles such that every rectangle is contained in P and the union of all rectangles is equal to P. In the following, for simplicity, polygons always denote rectilinear polygons and partitioning always denotes rectangular partitioning.

A *concave vertex* $v_1: (x_1, y_1)$ of P is a vertex having a 270° interior angle. A *reflex edge* of P is an edge connecting two concave vertices. Two concave vertices $v_1: (x_1, y_1)$ and $v_2: (x_2, y_2)$ which do not share the same edge of P are *co-grid* if they are co-horizontal ($y_1 = y_2$) or co-vertical ($x_1 = x_2$). A *chord* of P is a line segment contained in P connecting two cogrid vertices. If a rectilinear polygon contains no chords, then a minimal partition can be easily obtained by using the following principle.

1) For each concave vertex, select one of the edges. Note that there are two edges intersecting at each concave vertex.

2) Extend this edge until it hits another such extended edge or a boundary edge of P.

Throughout this paper, we shall assume that our simple polygons contain cords. Ferrari, Sankar, and Sklansky [3] showed that the size of a minimal partition is equal to $n - b + 1$ where b is the size of the largest set of nonintersecting chords. Consider Fig. 1(a). The set of chords is $\{ab, ef, gh, ij, ch, di\}$. The largest set of nonintersecting chords is $\{ab, ef, gh, ij\}$. Using these nonintersecting chords, a minimal partition can be constructed as shown in Fig. 1(b). Note that there might be other approaches to solve the minimal rectangular partion problem. However, this approach, which is based upon finding a largest set of nonintersecting chords will definitely lead to a minimal solution.

In [3], it was shown that the minimum partition of any simple polygon P containing chords can be found in six steps.

1) Find chords of P.

2) Construct a bipartite graph $B = (V, H, E)$ as follows. a) Each vertex v_i in V corresponds to a vertical chord i. b) Each vertex h_j in H corresponds to a horizontal chord j. c) Each edge $v_i h_j$ in E corresponds to the intersection of chords i and j.

3) Find a maximum matching [17] M of B.

4) Find a maximum independent set [17] S of B based on M. The nodes in S are not adjacent to each other and, therefore, the chords corresponding to nodes in S are nonintersecting chords. Denote the size of S as b.

5) Draw b nonintersecting chords corresponding to S to divide P into $b + 1$ subpolygons such that each subpolygon has no co-grid concave vertices.

6) Since each subpolygon contains no chords, a minimal partition of each subpolygon can be found by using the principle stated in the previous paragraph.

In [15] and [3], Hopcroft and Karp's algorithm [9] was used to find the maximum matching of a bipartite graph. Hopcroft and Karp's algorithm was designed for general bipartite graphs and runs in $O(n^{2.5})$ time where n is the number of vertices in the bipartite graph. Imai and Asano [10] proposed another algorithm to find the maximum matching without constructing the bipartite graph. Imai and Asano's algorithm runs in $O(n^{0.5}N)$ time where $N = \min\{m, n \log n\}$ and m is the number of edges in the bipartite graph. Imai and Asano's algorithm runs faster than Hopcroft and Karp's algorithm. However, Imai and Asano's algorithm is not the most suitable one for simple polygons (without holes). Some special properties of the chords of a simple polygon have not been explored.

As in [10], we shall not construct the bipartite graph. We shall make a detailed analysis of the properties of the chords of a simple polygon. Utilizing these special properties, we can have an efficient algorithm to find the maximum matching. Our algorithm requires $O(n \log \log n)$ time. After the maximum matching is found, we can find the maximum nonintersecting chords in linear time and, consequently, the partition problem for simple polygons can be solved in $O(n \log \log n)$ time.

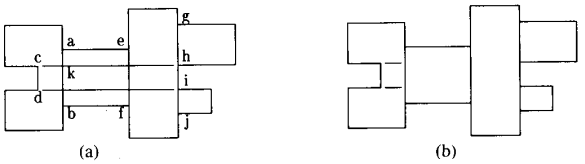

Fig. 1. Chords and the optimal partition.

III. Maximum Matching of a Special Bipartite Graph

In this section, we shall introduce a theorem which was initially discussed by Glover [6] and then generalized by Lipski and Preparata [12]. We shall also show that this theorem can be applied to the bipartite graph derived from the chords of a simple polygon.

Consider a bipartite graph $B = (V, H, E)$. We use $H(v_i)$ to denote the neighbors of a vertex v_1 in V and $V(h_g)$ to denote the neighbors of a vertex b_g in H.

Theorem 3-1 (Lipski and Preparata): Let $B = (V, H, E)$ be a bipartite graph. If $(v_i, h_g) \in E$ and $H(v_i) \subset H(v_j)$, for all $v_j \in V(h_g)$, then there is a maximum matching containing (v_i, h_g).

This theorem states the following. Let $v_i \in V$. Suppose that there is an $h_g \in H(v_i)$ satisfying the following property: for any $v_j \in V$, whenever $h_g \in H(v_j)$, it implies $H(v_i) \subset H(v_j)$, then there is a maximum matching containing (v_i, h_g). We define that $H_g(v_i) = \{h_g \mid h_g \in H(v_i)$ and $H(v_i) \subset H(v_j)$, for all $v_j \in V(h_g)\}$. By using Theorem 3-1, for any $h_g \in H_g(v_i)$, there is a maximum matching containing (v_i, h_g). For a general bipartite graph, it is possible that $H_g(v_i) = \phi$ for all v_i. Even if $H_g(v_i) \neq \phi$ for some v_i, it is not trivial to find a desired pair of v_i and $h_g \in H_g(v_i)$. However, for a bipartite graph derived from chords of a simple polygon, we shall prove that there always exists a vertical chord v_i such that $H_g(v_i) \neq \phi$ and we can find v_i and $h_g \in \mathrm{H}_g(v_i)$ efficiently. In the following, we shall define *left-free* and *right-convex* and show that, for any vertical chord v_i, if v_i is *left-free* and $H(v_i) \neq \phi$ is *right-convex*, then $H_g(v_i) \neq \phi$ and the horizontal chord $h_g \in H_g(v_i)$ where h_g has the *shortest right end* in $H(v_i)$.

Consider a vertical chord v_i of a simple polygon P. v_i slices the boundary of P into two parts. One part is left to v_i and the other part is right to v_i. We define that v_i is *left-free* if there is no other vertical chord whose both ends are on the left part. For any two vertical chords v_i and v_j, if v_i is left-free and $x(v_j) \leq x(v_i)$, then v_j must be higher than the upper end or lower than the lower end of v_i. Consequently, v_j does not intersect with any horizontal chord in $H(v_i)$.

Lemma 3-1: Let v_i be a left-free vertical cord. For any vertical chord v_j, if $x(v_j) \leqq x(v_i)$, then $H(v_i \cap H(v_j) = \phi$.

Proof: Immediately proved from the definition of left-free. Q.E.D.

The definition of right-convex is as follows. Let v_i be a vertical chord of a simple polygon. Let $h_1, h_2, \cdots, h_k$ be the horizontal cords of $H(v_i)$ sorted in the descending order of the y-position. Let $a_1, a_2, \cdots, a_k$ be the right ends of $h_1, h_2, \cdots, h_k$, respectively. If we start from a_1 to walk along the boundary of P in the clockwise direction, then the order of occurrences of right ends a_i's on the boundary is still in the sequence $[a_1, a_2, \cdots, a_k]$ because P is simply connected.

The *right-boundary* of $H(v_i)$ is a piece of the boundary of P, which starts from a_1 passing through all a_i, $1 < i < k$, and ends at a_k. $H(v_i)$ is *right-convex* if there is no vertical reflex edge on the right-boundary of $H(v_i)$. $H(v_i)$ is *right-concave* if it is not right-convex, i.e., there exists at least one vertical reflex edge on the right-boundary of $H(v_i)$.

The following lemma gives a sufficient condition for the existence of a vertical reflex edge on a piece of boundary. The proof of this lemma is not difficult but is very tedious. Therefore, we omit it.

Lemma 3-2: Let ΔP be a piece of the boundary of a simple polygon P such that the interior of P is on the left side of ΔP. Let a_1 and a_3 be two concave vertices and a_2 be any vertex on ΔP such that, if we traverse ΔP in clockwise direction, the occurrences of these three verticles on ΔP are in the sequence $[a_1, a_2, a_3]$ and the heights of them are $y(a_1) > y(a_2) > y(a_3)$. There is a vertical reflex edge between a_1 and a_3 on ΔP if one of the following conditions are true:

a) $x(a_1) > x(a_2)$ and $x(a_2) < x(a_3)$,
b) $x(a_1) = x(a_2)$ and $x(a_2) < x(a_3)$,
c) $x(a_1) > x(a_2)$ and $x(a_2) = x(a_3)$.

Lemma 3-3: Let $H(v_i)$ be right-convex. Then either the highest chord or the lowest chord in $H(v_i)$ has the shortest right end among all cords in $H(v_i)$.

Proof: Let $A = [a_1, a_2, \cdots, a_k]$ be the set of right ends of $H(v_1)$ sorted from high to low. Assume that neither the highest chord nor the lowest chord has the shortest right end. Assume that a_f, $1 < f < k$, is the shortest right end. We have $y(a_1) > y(a_f) > y(a_k)$. The right-boundary of $H(v_i)$ connects a_1, a_f, and a_k. The occurrences of a_1, a_f, and a_k on the right-boundary of $H(v_i)$ are in the sequence $[a_1, a_f, a_k]$. By Lemma 3-2, there exists a vertical reflex edge on the right-boundary of $H(v_i)$, a contradiction. Q.E.D.

Let v_i be a vertical chord such that $H(v_i)$ is right-convex. Consider a vertical chord v_j, $x(v_i) < x(v_j)$. We are going to prove in Lemma 3-4, which is the key lemma in the section, that if v_j intersects with the shortest horizontal chord in $H(v_i)$, then v_j intersects with every horizontal chord in $H(v_i)$. In order to prove Lemma 3-4, we first discuss some properties of the ends of v_j. Let h_g be the shortest chord in $H(v_i)$. Assume that v_j intersects with h_g. Let p_1 and p_2, respectively, be the upper end and the lower end of v_j. (See Fig. 2.) Consider the vertical edge ϵ of p_2. p_2 is the higher end of ϵ. Let p_2' be the lower end of ϵ. We have $y(p_2) > (p_2')$ as shown in Fig. 2(a). Then, consider the horizontal edge λ of p_2'. If p_2' is a convex vertex, then p_2' is the right end of λ and $x(p_2') > x(p_2'')$ where p_2'' is the left end of λ as shown in Fig. 2(b).

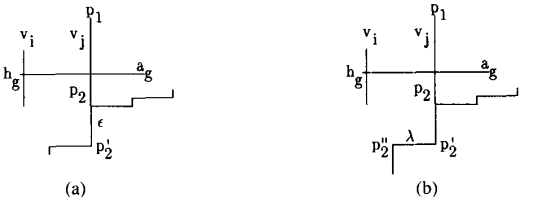

Fig. 2. Relative positions of p_1, p_2, and p_2'. (a) $y(p_2) > y(p_2')$. (b) p_2' is convex and $x(p_2') > x(p_2'')$.

Lemma 3-4: Let $H(v_i)$ be right-convex and h_g be the chord in $H(v_i)$ having the the smallest right end. For any vertical chord v_j, $x(v_j) > x(v_i)$, if v_j intersects with h_g, then v_j intersects with every $h_i \in H(v_i)$.

Proof: Assume that $h_f \in H(v_i)$ and h_f does not intersect with v_j. Let p_1 and p_2 be the upper end and the lower end of v_j, respectively. Let a_f be the right end of h_f. By Lemma 4-1, h_g is the highest chord or the lowest chord in $H(v_i)$. Without loss of generality, we assume that h_g is the highest chord in $H(v_i)$. Let ΔP denote the right-boundary of $H(v_i)$. Since v_j does not intersect with h_f, we have $y(p_2) > y(a_f)$ and, therefore, p_2 and a_f are concave vertices on ΔP. Let p_2' be the lower end of the vertical edge p_2. We have $x(p_2) = x(p_2')$. 1) If $p_2' = a_f$, then $\overline{p_2 a_f}$ is a vertical reflex edge on ΔP, a contradiction. 2) Assume that $a_f \neq p_2'$, we have $y(p_2) > y(p_2') > y(a_f)$. If p_2' is a concave vertex, then $\overline{p_2 p_2'}$ is a vertical reflex edge on ΔP, a contradiction. If p_2' is a concave vertex, then assuming that p_2'' is the vertex clockwise succeeding p_2' on ΔP, we have $x(p_2) > x(p_2'')$ and $x(a_f) > x(p_2'')$. Since $y(p_2) > y(p_2') = y(p_2'') > y(a_f)$, by Lemma 3-2, there is a vertical reflex edge between p_2 and a_f on ΔP, a contradiction. Q.E.D.

Based on Lemmas 3-3 and 3-4, we immediately have the following lemma.

Lemma 3-5: Let v_i be a vertical chord and $H(v_i)$ be right-convex. Let h_g be a horizontal chord with the smallest right end in $H(v_i)$. For any vertical chord v_j, $x(v_i) < x(v_j)$, if $h_g \in H(v_j)$, then $H(v_i) \subset H(v_j)$.

Lemma 3-1 and Lemma 3-5 show that, for a left-free vertical chord v_i of a simple polygon P, if $H(v_i) \neq \phi$ is right-convex, then $h_g \in H_g(v_i)$ where h_g has the shortest right end in $H(v_i)$. *Right-free* and *left-convex* can be defined in a symmetric manner as left-free and right-convex. If v_i is right-free and $H(v_i)$ is left-convex, then Lemma 3-1 to Lemma 3-5 are also true with suitable modifications. For simplicity, we neglect the proofs. We conclude our discussion in this section with the following theorem.

Theorem 3-2: Let v_i be a left-free (right-free) verical chord of a simple polygon P and $H(v_i) \neq \phi$ be right-convex (left-convex). Let $B = (V, H, E)$ be the bipartite graph derived from the chords of P. There exists a maximum matching M of B such that $v_i h_g$ is in M, where h_g is the chord with the shortest right (left) end in $H(v_i)$.

Proof: By Lemma 3-1 and 3-5, for any v_j, if $h_g \in H(v_j)$, then $H(v_i) \subset H(v_j)$. By Theorem 3-1, there is a maximum matching M such that $v_i h_g \in M$. Q.E.D.

IV. Finding Maximum Matching of a Horizontally Convex Polygon

In this section, we shall explain how to find the maximum matching of a horizontally convex polygon. The technique we illustrated will be later extended to simple polygons.

A simple polygon P is *horizontally convex* if, for any horizontal line segment, two ends of this line segment are contained in P implies that this line segment is contained in P. A *support edge* of P is an edge connecting two 90° angles. In a horizontally convex polygon, two horizontal support edges separate the boundary of the polygon into two chains of vertices, a *left chain* and a *right chain*. Consider a vertical reflex edge e_i and a vertical support edge s_i on the same chain. Assume that there is no other support or reflex edges between e_i and s_i on the same chain. For e_i and s_i on the left chain, a vertical chord v_k is *located* between e_i and s_i if $x(s_i) \leq x(v_k) \leqq x(e_i)$. For e_i and s_i on the right chain, a vertical chord is *located* between e_i and s_i if $x(s_j) > x(v_k) \geqq x(e_i)$. We can draw an extension through e_i such that the higher (lower) end of s_i is lower (higher) than the higher (lower) end of the extension. The *boundary shrinking* along this extension is the elimination of the boundary between e_i and s_i.

In general, for a given horizontally convex ploygon, we can start from the top support edge to trace the left chain and the right chain at the same time such that we are on the same heights at both chains. We keep on tracing until we find the first vertical reflex edge e_i. Assume that e_i is on the left chain. Let s_i be the last support edge traced before e_i on the same chain. For any vertical chord v_i between s_i and e_i, $H(v_i)$ is right-convex because there is no vertical reflex edge e_i' on the right-boundary of $H(v_i)$. If v_i is the left-most chord between s_i and e_i, then v_i is left-free. After v_i is matched and removed, the vertical chord succeeding v_i will be left-free. Therefore, for vertical chords v_i located between s_i and e_i, we can process them from left to right as follows.

1) If $H(v_i) = \phi$, then remove v_i from the polygon.
2) If $H(v_i) \neq \phi$, then v_i is matched with a horizontal chord $h_g \in H(v_i)$ where h_g is the chord with the shortest right end in $H(v_i)$. After matching, v_i and h_g are removed from the polygon.

After all vertical chords between s_i and e_i have been processed as described above, s_i and e_i are eliminated by the boundary shrinking along the vertical extension through e_i. After boundary shrinking, e_i and s_i do not exist in the new polygon. If e_i and s_i are on the right chain, we will process vertical chords, v_i located between e_i and s_i from right to left.

Repeatedly applying the above procedure, we can eliminate all vertical reflex edges on both chains. When all vertical reflex edges are eliminated, the remaining boundary forms a polygon with no reflex edges (a convex polygon) and, therefore, the neighbors of vertical chords of the remaining polygon are both right-convex and left-convex, which can be processed from left to right or from right to left.

A horizontally convex polygon can also be processed from the bottom support edge. In this case, when we trace upwards along the left chain and right chain, the vertical reflex edge with the lowest lower end point will be eliminated first. It is important to note that, if we process a horizontally convex polygon from the top to the bottom, we always execute boundary shrinkings along the *upward extensions* of vertical reflex edges. If we process from the bottom to the top, then boundary shrinkings are executed along the *downward extensions* of vertical reflex edges. The following algorithm, Algorithm 1, implements the above ideas.

Algorithm 1

input: A horizontally convex polygon P.
output: The maximum set M of matched chords of P.
Steps:

1) Find horizontal chords and vertical chords of P;
2) Trace the left chain and the right chain of P at the same time to find the first pair of support edge s_i and reflex edge e_i of P;
3) If there is no vertical cords located between s_i and e_i, then execute boundary shrinking along e_i; otherwise, find the vertical chord v_i located beween s_i and e_i such that there is no other vertical located between s_i and v_i;
4) Match v_i with the shortest horizontal chord h_g in $H(v_i)$;
5) Put $v_i h_g$ into M;
6) Remove v_i and h_g from P to have a new polygon P';
7) Recursively use Algorithm 1 to find M' for P';
8) $M = M \cup M'$;
9) end;

Theorem 4-1: Algorithm 1 finds the maximum set of matched chords of a horizontally convex polygon P.

Proof: Let v_i be a vertical chord being processed by Algorithm 1. Algorithm 1 ensures that v_1 is left-free and $H(v_1)$ is right-convex. In Algorithm 1, v_1 is matched with $h_g \in H(v_1)$ where h_g has the shortest right end in $H(v_1)$. By Theorem 3-2, there exists a maximum matching of P, which contains $v_1 h_g$. After v_1 and h_g are matched, we move the x-position of the upper end of v_1 a small dis-

tance (and modify the x-positions of the other relevant vertices, suitably,) such that v_1 will not exist and no other vertical chords will be produced. We also move the left end of h_g a small distance in the y-direction such that h_g will not exist and no other horizontal chords will be produced. The resulting polygon is still a horizontal convex polygon without chords v_1 and h_g. All other chords remain unchanged. Let B^1 and B^2 be the bipartite graph derived from chords of the old polygon and the new polygon, respectively. $B^1 - \{v_1, h_g\} = B^2$. Let M^2 be the maximum matching of B^2. By Theorem 3-1, the maximum matching M^1 of B^1 is equal to $M^2 \cup \{v_1 h_g\}$. We can recursively apply the same procedure to the new polygon to find M^2. Therefore, Algorithm 1 finds the maximum set of matched chords of P. Q.E.D.

V. Finding the Maximum Matching of a Simple Polygon

This section includes two subsections. In Section V-5.1, we will explain our basic ideas for finding the maximum matching for a simple polygon and prove that our ideas are correct. In Section V-5.2, we will explain our algorithm in detail and analyze the time required for executing our algorithm.

5.1. Basic Ideas

Before presenting the many definitions needed by our algorithm, let us now present a high level informal description of our approach. Our approach consists of the following steps.

1) We find all of the horizontal reflex edges of the given simple polygon. For instance, in Fig. 3, the horizontal reflex edges are $\{\boldsymbol{ab}, \boldsymbol{cd}\}$.

2) Through ***ab*** and ***cd***, draw horizontal extensions. These horizontal extensions will decompose this simple polygon into five horizontally convex subpolygons as shown in Fig. 3.

3) Construct a tree of these horizontally convex subpolygons by the following rule: two horizontally convex subpolygons are connected if and only if they share one horizontal extension through some horizontal reflex edge. For the case shown in Fig. 3, the tree is shown in Fig. 4.

4) Select any node as the root. For instance, for the tree in Fig. 4, we may select 5 as the root. We then use a post-order [1] sequence to process the nodes. Again, for the tree in Fig. 4, a post-order sequence is 1, 2, 4, 3, 5.

5) For each node under processing, we use the method described in Section IV to find vertical reflex edges and the vertical chords relevant to these vertical reflex edges. A vertical reflex edge is elminated by the boundary shrinking after its relevant vertical chords are matched. For example, subpolygon 1 has two vertical reflex edges $\{\boldsymbol{hg}, \boldsymbol{fe}\}$. The vertical chords relevant to ***fe*** are v_1 and v_2 and vertical chords relevant to ***hg*** are v_3 and v_4, as shown in Fig. 5(a). With respect to ***hg*** and ***fe***, the matching found is $\{(v_1, h_2), (v_2, h_1), (v_3, h_3), (v_4, h_4)\}$, and the resulting subpolygon after boundary shrinkings is shown in Fig. 5(b).

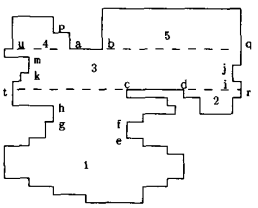

Fig. 3. A simple polygon.

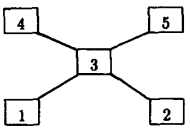

Fig. 4. The tree of subpolygons.

Fig. 5. Processing subpolygon 1.

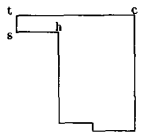

Fig. 6. Subpolygon 1 after processing.

6) After all vertical reflex edges are eliminated, consider two ends of the upper support edge ***tc*** of subpolygon 1. The vertical edges at each end of ***tc*** are viewed as vertical reflex edges, i.e., ***ts*** and ***cv*** are viewed as vertical reflex edges and should be eliminated by boundary shrinkings. There is no boundary left to ***ts*** and no boundary shrinking is needed at ***ts***. However, the boundary right to ***cv*** should be eliminated and the relevant vertical chord v_5 should be matched. The matching found in this step is $\{(v_5, h_5)\}$ and the resulting subpolygon is shown in Fig. 6.

7) We merge the resulting subpolygon 1 to its parent node, namely, subpolygon 3, such that subpolygon 3 becomes that in Fig. 7.

8) We repeat the same procedure illustrated in steps 5), 6), and 7) to all other subpolygons in the post-order sequence.

In step 5), it is stated that we can use the method described in Section IV to eliminate vertical reflex edges.

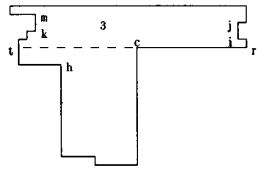

Fig. 7. Merging subpolygon 1 to subpolygon 3.

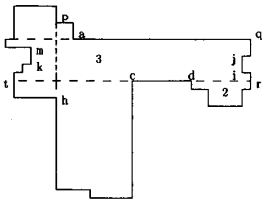

Fig. 8. Subpolygon 3 after merging with its children.

However, there is a little difference. In Section IV, we have shown that, for a horizontally convex polygon P, P can be processed from the top to the bottom or from the bottom to the top. However, in this section, any subpolygon should be processed in a predefined direction as explained below. For a node n_i of the resulting tree in step 3), there are two support edges of n_i. We define that the *master bound* of n_i is the support edge (the horizontal extension) between n_i and its parent. If n_i is the root, then we assign the upper support edge of n_i to be the master bound of n_i. (Note that it makes no difference to assign the lower support edge as the master bound for the root.) The *slave bound* is the other support edge of n_i. The master bound and the slave bound of a node are fixed. For example, in Fig. 3, the master bound of subpolygon 3 is ***bq*** and the slave bound is ***tr***. This will not be changed even after subpolygons 1, 2, and 4 are merged to subpolygon 3. The processing direction of a leave node, which is always a horizontally convex subpolygon, of the tree is from the slave bound toward the master bound. However, the internal nodes may not be a horizontally convex subpolygon after merging. For example, the resulting subpolygon 3 after merging with subpolygons 1, 2, and 4 is not a horizontally convex subpolygon because the horizontal reflex edge ***cd*** exists as shown in Fig. 8. However, we are sure that, if there exist vertical reflex edges in an internal node n_i after merging, these vertical reflex edges must be on the left chain and the right chain between the master bound and the slave bound of n_i because all vertical reflex edges of the child nodes of n_i have been eliminated by boundary shrinkings. For example, in Fig. 8, the vertical reflex edges ***ji*** and ***mk*** are on the boundary between the master bound ***bq*** and the slave bound ***tr***. We still can find the vertical reflex edges and compare their heights. We will prove in Theorem 5-1 that the matching algorithm, Algorithm 1 of Section IV, still finds the maximum matching for the subpolygons of this section.

Steps 6) and 7) need some explanations. For the given simple polygon, there might exist vertical chords intersecting with the horizontal extensions drawn through the horizontal reflex edges. Consider Fig. 3. There is a vertical chord ***ph*** crossing the horizontal extensions between subpolygons 1 and 3 and subpolygons 3 and 4. When we process subpolygon 1 or subpolygon 4, we cannot find ***ph***. ***ph*** can be found in subpolygon 3 only after subpolygon 1 and subpolygon 4 are merged to subpolygon 3. In order to solve this anomaly, we define *vertically visible* for a concave vertex v and a horizontal edge ϵ. v is *vertically visible* to ϵ if we draw a vertical extension through v and ϵ is the first hit edge. If v is vertically visible to the master bound, then it is possible that there exists a vertical chord through v and this chord cannot be found in n_i. When we process node n_i, we do not process the vertices which are vertically visible to the master bound of n_i. In other words, for any vertical chord v_i of n_i, if v_i does not intersect with the master bound of n_i (v_i can intersect at two ends of the master bound), v_i will be processed in n_i. In step 6), the vertical edges at each end of the master bound are viewed as vertical reflex edges. We call them *virtual vertical reflex edges*. After all relevant vertical chords are processed, we execute boundary shrinkings to eliminate these virtual vertical reflex edges and to eliminate all vertices in n_i which are invisible to the master bound. The remaining boundary of n_i, which only contains the vertically visible vertices, will be merged to the parent of n_i for further processing.

In order to have the intersecting conditions of the chords in n_i to be the same as in the original polygon, we can imagine that there are concave vertices on the master bound such that the vertical chords through the visible vertices exist in n_i. Adding concave vertices on the master bound will not influence our matching procedure for n_i because the vertices visible to the master bound will not be processed in n_i. Though we do not actually draw concave vertices on the bound of n_i, we can accept the argument that the intersecting conditions of chords in n_i are the same as in the original polygon. Consequently, the result of processing subpolygon n_i is applicable to the original polygon.

The last detail that we have to consider is that, when we decompose a given simple polygon into subpolygons, the horizontal extension through horizontal reflex edges might be horizontal chords. For example, in Fig. 3, the extension ***ua*** between subpolygons 3 and 4 is a horizontal chord. Since each extension defines a parent node and a child node, if it is a horizontal chord, then it will be recorded as a horizontal chord in the child node. For example, ***ua*** will be a horizontal chord of subpolygon 4. Algorithm 2 implements the above ideas.

Algorithm 2

input: A simple polygon P.

output: The maximum set M of matched chords of P.

steps:

1) partition p into horizontally convex subpolygons and construct a tree T of horizontally convex subpolygons;

2) arbitrarily assign a node of T as the root and determine the master bound and the slave bound for each node;
3) visit nodes n_i of T in post-order and do the following for n_i
   assign two vertical edges at two ends of the master bound of n_i to be two vertical reflex edges of n_i;
   use Algorithm 1 to find a matching M_i of n_i and to eliminate all vertical reflex edges of n_i;
   /*Note that Algorithm 1 will trace from the slave bound to find vertical reflex edges but will trace the whole boundary to find the relevant vertical support edges and chords.*/
   $M = M \cup M_1$;
   if n_i is the root, then
      return;
   else
      merge n_i to its parent;
   end if;
end;

Theorem 5-1: Algorithm 2 finds the maximum set of matched chords of a simple polygon P.

Proof: Consider step 3) of Algorithm 2. The vertical reflex edges of n_i are found by tracing the left chain and the right chain from the slave bound. Without loss of generality, we assume that the slave bound of n_i is lower than the master bound and, consequently, the slave bound is lower than any vertical reflex edge of n_i. Let e_i be the first found vertical reflex edge. We are sure that there is no other vertical reflex edge lower than e_i. Assume that e_i is on the left chain. After e_i is found, we can find the corresponding support edge s_i of e_i. s_i might not be on the boundary between the slave bound and the master bound but s_i is always lower than e_i. Therefore, for any vertical chords v_i located between e_i and s_i, $H(v_i)$ is right-convex in n_i because there is no vertical reflex edge lower than e_i. Lemma 3-1 to Lemma 3-5 are still valid for v_i. Therefore, the vertical chords located between s_i and e_i can be processed from left to right to find their matches. Using the similar techniques in the proof of Theorem 4-1, we can prove that Algorithm 2 finds the maximum matching of P. Q.E.D.

It is important to note that although some vertices might be recursively merged to their parent nodes, we are sure that any vertex will only be *processed constant times* for the following reasons. Consider an internal node n_i of T. In Algorithm 2, we only trace the boundary between the master bound and the slave bound to find the lowest or the highest vertical reflex edge of n_i. After a desired vertical reflex edge is found, we will trace the whole boundary of n_i to find the relevant vertical support edge and the relevant vertical chords. However, the traced boundary will be eliminated by the boundary shrinking along this vertical reflex edge. Therefore, any vertex will only be traced constant times before being eliminated. It should also be noted that in Algorithm 2, the ends of a master bound are treated the same as ends of vertical reflex edges and the vertices of a subpolygon n_i, which are vertically visible to the master bound of n_i, are not processed in n_i. In order to execute Algorithm 2 efficiently, it will be modified in Section V-5.2 to find a maximum matching in $O(n \log\log n)$ time.

5.2. Algorithms

For a given simple polygon P, the algorithm for finding the maximum matching is shown in Algorithm MAIN. In addition to finding the maximum matching M of P, MAIN also outputs the set U of *unmatched chords* with respect to M. M and U will be used in Section VI to find the maximum set of nonintersecting chords of P in linear time, which is the final goal of this paper.

Algorithm MAIN

input: A simple polygon P.
output: The maximum matching M of chords of P and the set U of unmatched chords with respect to M.

begin
1) Partition P into horizontally convex subpolygons.
   Construct a tree T_h of horizontally convex subpolygons;
   For each node n_k of T_h, find horizontal chords in n_k;
2) Construct a tree T_v storing the x-positions of vertical reflex edges and end points of master bounds;
3) Visit nodes of T_h in post-order;
4) For each visited node n_k of T_h, do
   Find the maximum matching of cords in n_k;
   If n_k is not the root of T_h, then merge n_k to the parent of n_k;

end

We discuss each step in detail as follows.

Step 1: The input polygon P is a sequence of vertices. We store P into an array $R(P)$. For each vertex v of P, there are two data items about v. One is the position of v on the plane and the other is that v is the ith input element of P. For the ith vertex v, we can locate v in $R(P)$ in constant time. In order to partition P into horizontally convex subpolygons, we have to draw horizontal extensions through horizontal reflex edges to hit the nearest boundary. Tarjan and Van Wyk [18] have defined a *vertex-edge visible pair* to be a vertex and an edge that can be connected by an open horizontal line segment that lies entirely inside P. For a particular concave vertex v, if the vertex-edge visible pairs of v is known, the hit point of v can be computed in constant time. In [18], Tarjan and Van Wyk proposed an $O(n \log\log n)$ algorithm to find all the vertex-edge visible pairs. We use Tarjan and Van Wyk's algorithm to find all the vertex-edge visible pairs as preprocessing. After these pairs are found, we record this information back into $R(P)$ such that, for the ith vertex v of P, we can decide the hit point of v in constant time.

In addition to $R(P)$, P is also stored in *finger search*

tree [18] which can be constructed in linear time. Using a concave vertex v and its hit point hit(v), the finger search tree of P is partitioned into two subtrees (subpolygons): one contains the vertices from v to hit(v) and the other contains rest vertices. This is a *three-way splitting* [18]. Each three-way splitting will reduce one end of horizontal reflex edges. For each subpolygon, we can recursively apply the three-way splitting until there exists no end of horizontal reflex edges. If there are n vertices in the original finger search tree and i vertices from v to hit(v) (hit(v) is exclusive), it takes $O(1 + \log(\min\{i, n - i\} + 1))$ amortized time [18] to split this finger search tree into two subtrees containing i and $n - i$ verticles, respectively. (Note that, though the hit point will be a vertex in each subpolygon, we do not include these two new vertices in the subtrees because the hit points are irrelevant for the further partitioning.) Let $T(n)$ be the worst-case total time required for partitioning. We have the following recurrence formula:

$$T(n) = \begin{cases} O(1) & \text{if } n \le n_0 \text{ where } n_0 \text{ is a constant;} \\ \max\limits_{1 \le k < n} \{T(i) + T(n - i) \\ \quad + O(1 + \log(\min\{i, n - i\} + 1)) \\ \quad \text{if } n > n_0. \end{cases}$$

Solving this recurrence, we have $T(n) = O(n)$ [13]. Therefore, the total time required for partitioning P into horizontally convex subpolygons is $O(n \log\log n)$. Note that it may take $O(\log n)$ time to split P into two subpolygons without using the finger search tree and, consequently, in the worst case, it may take $O(n \log n)$ time to split P into horizontally convex subpolygons. However, the finger search tree ensures us that we can split P into horizontally convex subpolygons in $O(n)$ time.

The tree T_h of horizontally convex subpolygons is constructed according to the rule described in Section IV-4.4. Each node of T_h corresponds to a finger search subtree storing the vertices of a horizontally convex subpolygon, i.e., there is a pointer in this node pointing to the subtree. T_h can be constructed during the process of the partitioning P into horizontally convex subpolygons. Initially, there is only one node in T_h corresponding to the original finger search tree. Whenever the finger search tree or a subfinger search tree is split, the corresponding node in T_h will also be split and the resulting nodes are connected. T_h is constructed when P is partitioned into horizontally convex subpolygons. After constructing T_h, arbitrarily assign a node of T_h as the root. The post-order of nodes in T_h is the processing sequence of horizontally convex subpolygons. In the post-order sequence, a node will be processed if all its children have been processed.

Horizontal chords can be found in linear time by tracing the left-chain and the right chain of a node n_k of T_h. When two concave vertices on different chains of n_k are found to have the same y-position, a horizontal chord exists. When a horizontal chord h is found, we record the existence of h at both ends of h on the boundary of n_k.

Step 2: In Section V 5.1, we have explained that horizontal chords will be shortened after boundary shrinkings. Boundary shrinkings are executed along the vertical extensions through vertical reflex edges and virtual vertical reflex edges. These extensions are fixed after the master bound of each node in T_h is determined. These extensions partition P into a set of vertically convex subpolygons. Tree T_v is constructed as follows: there is a node in T_v for each vertically convex subpolygon and an edge for each extension. Among the nodes of T_v, there is a node containing the master bound of the root of T_h. We assign this node as the root of T_v. When a boundary shrinking is executed along an extension, the relevant horizontal chords in the child part are shortened to the x-position of this extension. Therefore, finding new positions of horizontal chords can be executed on T_v using the *static tree set union* algorithm of Gabow and Tarjan [5]. In [5], for static tree set union, a rooted tree with k nodes is given. Each node of this tree is a singleton set. The LINK(v) operation is to unite a node v in the tree to its parent. (Actually, LINK(v) is to make a mark on node v.) FIND(v) will return v if node v is not marked by LINK; otherwise, the nearest unmarked ancestor of v will be returned [5]. We can apply the static tree set union to tree T_v. T_v corresponds to the given rooted tree. The boundary shrinking along an extension σ corresponds to making a mark on the child node of the edge defined by σ. Since the child node v of σ is uniquely defined, LINK(v) is well defined for the boundary shrinking along σ. Our FIND operation is executed as follows. Let p be an end point of a horizontal chord h. Assume that p is originally contained in the node v of T_v. When v is marked by LINK(v), it means that p has been shortened to the position of the edge connection v and its parent. If p is recursively shortened, the new position of p is the edge incident to the nearest unmarked ancestor of v. Therefore, FIND(v) which returns the nearest unmarked ancestor of v can be used to find the new positions of p which is originally contained in v. It is shown in [5] that a sequence of $O(m)$ intermixed LINK and FIND operations can be executed in $O(m + k)$ time where k is the number of nodes in the static tree. Since the number of vertical reflex edges and master bounds is of $O(n)$, k is bounded by $O(n)$. The number of LINK operations is at most equal to the number of k. The number of FIND operations is equal to the number of vertical chords which is also bound by $O(n)$. Therefore, $O(m + k)$ is bounded by $O(n)$.

Step 3: The ideas for finding maximum matching of a node of T_h have been explained in Section V-5.1. Assume that a vertical reflex edge e_i (or an end of a master bound) and its corresponding vertical support edge s_i have been found. We only explain how the vertical chords v_i between e_i and s_i and their neighbors $H(v_i)$ are found.

Assume that e_i and s_i are on the left chain. The vertical chords between e_i and s_i can be found as follows. s_i partitions the left chain into two parts: the *upper chain* and the *lower chain*. Starting from s_i and tracing the upper chain and the lower chain of s_i at the same time, a vertical

chord exists when two concave vertices on the different chains have the same x-position. (Note that the vertices which are visible to the master bound of n_k are not traced and vertical chords crossing the master bound of n_k will not be found.) For a found vertical chord v_i, $H(v_i)$ can be found as follows. 1) Assume that v_i is the first vertical chord with respect to s_i. $H(v_i)$ is the set of horizontal edges whose one end is on the boundary between v_i and s_i and the other end is not. $H(v_i)$ can be found by tracing the boundary from s_i to v_i. 2) Assume that v_i is not the first vertical chord and v_{i-1} is the vertical chord prior to v_i. $H(v_i)$ is equal to the union of $H(v_{i-1})$ and the set of horizontal chords found on the boundary between v_{i-1} and v_i. For simplicity, we set $s_i = v_0$ and $H(v_0) = \phi$. Procedure FIND_NEIGHBOR(v_{i-1}, v_i) will return $H(v_i)$ where v_i is the vertical next to v_{i-1}. In FIND_NEIGHBOR, a double linked list $H(v_i)$ is constructed. Two pointers, head$(H(v_i))$ and tail$(H(v_i))$, point to the head and the tail of $H(v_i)$, respectively. Since we assume that e_i and s_i are on the left-chain, we trace the upper chain and the lower chain of s_i from left to right. If the left end of a horizontal chord h is found on the upper chain, then h will be put at head$(H(v_i))$. If the left end of h is found on the lower chain, then h will be put at tail$(H(v_i))$. If the right end of h is found on the upper chain, then head$(H(v_i))$ will be removed and h will be set to *unmatched*. If the right end of h is found on the lower chain, then tail$(H(v_i))$ will be removed and h will be set to be *unmatched. The unmatched chords are put into a set U.* U will be used to find the maximum independent set. The removed horizontal chord is always on the head or the tail of $H(v_i)$ because horizontal chords never intersect with each other. For a horizontal chord $h \in H(v_i)$, assume that the left end of h is on the upper (lower) chain. It is possible that the right end of h is found on the lower (upper) chain. However, at the time the right end of h is traced by FIND_NEIGHBOR, h is the tail (head) of the list $H(v_i)$.

Algorithm FIND_NEIGHBOR(v_{i-1},v_i)
input: Two vertical chords v_{i-1} and v_i, where v_{i-1} and v_i are on the left-chain, $x(v_{i-1}) < x(v_i)$, and v_{i-1} has been processed.
output: $H(v_i)$
begin
/*Assume that v_{i-1} is left to v_i and is on the left chain. Similar codes can be written for v_{i-1} and v_i on the right chain. For simplicity, we neglect the codes for the right chain.*/
make an empty list $H(v_i)$;
$H(v_i) = H(v_{i-1})$;
trace the upper chain and the lower chain of vertices between v_{i-1} and v_i;
for each found concave vertex p_i, do:
case A: p_i is the left end of a horizontal chord h, then
subcase 1: h is matched, then skip;
subcase 2: p_i is on the upper chain, then put p_i at head$(H(v_i))$;
subcase 3: p_i is on the lower chain, then put p_i at tail$(H(v_i))$;
case B p_i is the right end of a horizontal chord h, then
subcase 1: h is matched, then skip;
subcase 2: p_i is on the upper chain, then delete head$(H(v_i))$;
subcase 3: if p_i is on the lower chain, then delete tail$(H(v_i))$;
if h is not matched, then put h into U;
end do;
end.

In FIND_NEIGHBOR, since $H(v_{i-1})$ is directly assigned to $H(v_i)$, the horizontal chords common to $H(v_i)$ and $H(v_{i-1})$ are not to be found again. Therefore, the vertices on the boundary left to v_{i-1} do not have to be traced for $H(v_i)$ and the time required for finding $H(v_i)$ is proportional to the number of vertices on the upper chain and the lower chain between v_{i-1} and v_i. The total time required for Algorithm MAIN to execute FIND_NEIGHBOR to find $H(v_i)$ for all v_i is $O(n)$. The following property is obvious from Algorithm FIND_NEIGHBOR. This property will be used for finding the maximum independent set. Let p_i and p_j be two vertices. Since FIND_NEIGHBOR traces every vertex in a fixed sequence. p_i is *traced by FIND_NEIGHBOR before* p_j if p_i is prior to p_j in the tracing sequence.

Property 5-1: Let p_i be a concave vertex between a pair of neighboring support edge and vertical reflex edge (or between a support edge and an end of a master bound). Let hit(p_i) be the vertical hit point of p_i. The vertical line segment $\overline{p_i \text{hit}(p_i)}$ slices the polygon into two pieces. Then, either the vertices on the left-piece or the vertices on the right-piece of $\overline{p_i \text{hit}(p_i)}$ are traced by FIND_NEIGHBOR before p_i.

The maximum number of matched chords in n_k can be found by procedure MATCHING. The basic ideas have been explained in Section V-5.1. In MATCHING, a vertical chord v_i whose neighbor is empty will be set to *unmatched* and be put into U. Otherwise, v_i will be matched with the head or the tail of the list $H(v_i)$.

Algorithm MATCHING
input: A node n_k of tree T_h
output: M: the maximum set of matched chords of n_k.
U: the set of unmatched chords.
begin
1) assign the vertical edge at each end of the master bound of n_k as vertical reflex edges;
2) trace from the slave bound of n_k to find the nearest vertical reflex edge e_i and its corresponding vertical support edge s_i.
/*Assume that e_i and s_i are on the left chain.*/
3) trace from s_i to find vertical chords between e_i and s_i;

do for vertical chords v_i between e_i and s_i from left to right.
FIND_NEIGHBOR(v_{i-1},v_i);
if $H(v_i)=\phi$, then $U=U\cup\{v_i\}$;
x_1=FIND(head($H(v_i)$));
x_2=FIND(tail($H(v_i)$));
if $x_1<x_2$, then h_g=head($H(v_i)$);
else h_g=tail($H(v_i)$);
put v_ih_g into M;
remove v_i and h_g;
end do;
execute boundary shrinking along e_i;

4) repeat step 2 and step 3 until there is no vertical reflex edge;
5) if n_k is not the root, then LINK(n_k);
else $U=U\cup H(v_i)$; /*Remaining horizontal chords are unmatched chords.*

end;

In MAIN, the time required for partitioning P into horizontally convex subpolygons is $O(n \log\log n)$ where n is the number of vertices of P. The time required for processing each node of t_h is proportional to the number of vertices of this node because each vertex in this node is traced constant times. It takes totally $O(n)$ time to process all nodes of T_h. Therefore, the total time required for finding the maximum matching of P is $O(n \log\log n)$.

Theorem 5-2: The total time required for Algorithm MAIN to find the maximum matching of chords of a simple polygon P is $O(n \log\log n)$ where n is the number of vertices of P.

As for a horizontally (vertically) convex polygon or a convex polygon, it takes $O(n)$ time to find the maximum matching of chords because we do not have to partition it into subpolygons. This time bound is an obvious lower bound.

Theorem 5-3: Algorithm MAIN is optimal for finding the maximum matching of chords of a horizontally (vertically) convex polygon.

VI. Finding Maximum Nonintersecting Chords

In this section, we will show that, without constructing the bipartite graph of chords, the maximum independent (nonintersecting) set of chords can be found in linear time based on the matched pairs and unmatched chords found in Section V. We first give an example to explain our approach and, then, prove that our approach is correct. At the end of this section, we will give an Algorithm which runs in linear time.

Consider Fig. 9. In Fig. 9, a simple polygon P with one horizontal reflex edge ***uv*** is shown. In order to find the maximum matching, as explained in Section V, P should be partitioned into horizontally convex subpolygons by drawing horizontal extensions through ***uv*** and a tree of subpolygons should be constructed. Assume that ***wa*** is the master bound of the root of this tree. For any horizontal or vertical cord u of P, u slices the boundary

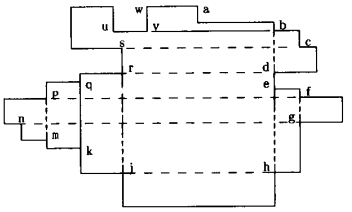

Fig. 9. A simple polygon with its matched chords.

of P into two pieces. One piece contains the master bound ***wa*** and this piece is the *main boundary* of u, denoted as $mb(u)$. The other piece is the *second boundary* of u, denoted as $sb(u)$. For example, consider the vertical chord ***qk***. $mb(\boldsymbol{qk})$ is the piece of boundary right to ***qk*** and $sb(\boldsymbol{qk})$ is the piece of boundary left to ***qk***. As for the horizontal chord ***vb***, $mb(\boldsymbol{vb})$ is the boundary above ***vb*** and $sb(\boldsymbol{vb})$ is the boundary below ***vb***. In Fig. 9, if we use Algorithm MAIN to find the maximum matching, the found maximum matching is $M = \{(\boldsymbol{fg}, \boldsymbol{pf}), (\boldsymbol{eh}, \boldsymbol{jh}), (\boldsymbol{pm}, \boldsymbol{ng}), (\boldsymbol{rj}, \boldsymbol{rd}), (\boldsymbol{bd}, \boldsymbol{sc})\}$, and the set of unmatched chords is $U = \{\boldsymbol{qk}, \boldsymbol{vb}\}$ where ***qk*** is vertical and ***vb*** is horizontal. Note that the elements in M and U are ordered according to their outputting sequence. The maximum independent set can be found by the following steps.

1) Let S be the set of maximum independent set. Initially, $S = \phi$.

2) Sort unmatched chords in U according to their outputting sequence. In our example, the outputting sequence of U is ***qk***, ***vb***.

3) Take the first element out from U and put it into S. That is, $S = \{\boldsymbol{qk}\}$.

4) Since ***qk*** is vertical, put the vertical chords which are on $sb(\boldsymbol{qk})$ into S. There is only one vertical chord ***pm*** on $sb(\boldsymbol{qk})$. Therefore, $S = \{\boldsymbol{qk}, \boldsymbol{pm}\}$.

5) Eliminate $sb(\boldsymbol{qk})$ to have a new polygon as shown in Fig. 10. Note that horizontal chords ***pf*** and ***ng*** do not exist in the new polygon. ***ng*** is matched with ***pm*** which has been put into S. ***pf*** is matched with ***fg*** which is an unmatched chord in the new polygon. We put ***fg*** into a set W, i.e., $W = \{\boldsymbol{fg}\}$. W is the set of unmatched chords which are caused by eliminating $sb(\boldsymbol{qk})$.

6) Take an element from W, i.e., ***fg***, and put ***fg*** into S. $S = \{\boldsymbol{qk}, \boldsymbol{pm}, \boldsymbol{fg}\}$. Since there is no vertical chord on $sb(\boldsymbol{fg})$, no chords will put into S together with ***fg***.

7) Eliminate $sb(\boldsymbol{fg})$. The resulting polygon is shown in Fig. 11. Since no horizontal chords are eliminated together with $sb(\boldsymbol{fg})$, no unmatched chords are produced in the new polygon and, therefore, no chords will be added into W.

8) If W is not empty, then repeat steps 6) and 7). Since W is empty now, we execute the next step.

9) Take the first element, ***vb***, from U and repeat steps 4)–8). That is, we first put ***vb*** into S. Since ***vb*** is horizontal, we put all horizontal chords, ***jh***, ***rd***, and ***sc***, on $sb(\boldsymbol{vb})$ into S. Then, we eliminate $sb(\boldsymbol{vb})$ as shown in Fig. 12. Vertical chords ***eh***, ***rj***, and ***bd*** are eliminated also. Since the horizontal chords matched with these three vertical

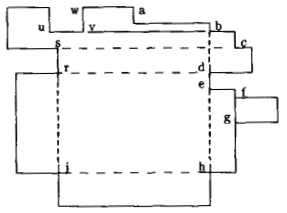

Fig. 10. Eliminating $sb(gk)$.

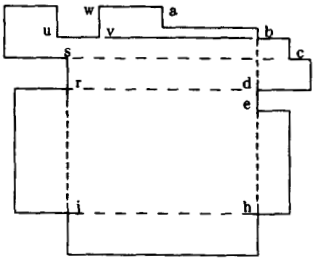

Fig. 11. Eliminating $sb(fg)$.

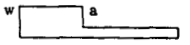

Fig. 12. Eliminating $sb(vb)$.

chords have been put into S, no horizontal chords will be added into W.

10) Since U and W are empty, the whole process stops. We have $S = \{qk, pm, fg, vb, jh, rd, sc\}$.

Two chords have the same *orientation* if they are both vertical and both horizontal. *orient(u)* is a set of chords such that, for any $t \in \text{orient}(u)$, t is on $sb(u)$ and t has the same orientation as u. For any chord u, $sb(u)$ is *consistent* if every unmatched chord u' whose both ends are on $sb(u)$ has the same orientation as u, i.e., $u' \in \text{orient}(u)$. In order to prove that the above operations correctly find the maximum matching, we will prove the following.

a) Let M be *any* maximal matching of a simple polygon P and let U be the set of unmatched chords with respect to M. We prove that, for any $u \in U$, if $sb(u)$ is consistent, then u and orient(u) are in the maximum independent S of P. This will be proved in Corollary 6-1 and Lemma 6-1. In addition, we show in Lemma 6-5 that, $sb(u_1)$ is consistent where u_1 is the first unmatched chord output by Algorithm MAIN.

b) After eliminating $sb(u_1)$, we have a new polygon. Let S' be a maximal independent set of the new polygon. We prove that $S = \{u_1\} \cup \text{orient}(u_1) \cup S'$. This is proved in Lemma 6-3. This lemma tells us that S can be found by recursively finding S'.

c) In order to find S', we first try to find a maximal matching and the relevant unmatched chords of the new polygon. In Lemma 6-2, we prove that, after eliminating $sb(u_1)$, the set of remaining matched chords is a maximal matching of the new polygon. That is, $M_1 = M - \{\text{chords on } sb(u_1)\}$ is a maximal matching of the new polygon. Also in Lemma 6-2, we prove that $W_1 \cup U - (\{u_1\} \cup \text{orient}(u_1))$ is the set of unmatched chords with respect to M_1 where W_1 is the set of unmatched chords caused by deleting $sb(u_1)$.

d) We show that, if order elements of $U - (\{u_1\} \cup \text{orient}(u_1))$ according to their outputting sequence and put W_1 at the beginning of $U - (\{u_1\} \cup \text{orient}(u_1))$ to have a new sequence $U' = [w_1, \cdots w_k, u_2, \cdots, u_n]$, then $sb(u_f)$ is consistent where u_f is the first element in U'. (Note that $u_f = u_2$ if $W = \phi$.) This is proved in Proposition 6-1. By Lemma 6-1, we can put u_f and orient(u_f) into S'.

In the following, we will use several terms of bipartite graphs such as maximum matching, maximum independent set, minimum node cover, alternating path and augmenting path in our theorems and lemmas without any explanation which can be found in most text books such as [17]. Let $B = (V, H, E)$ be a bipartite graph, M be *any* maximum matching of B, U be the set of unmatched nodes with respect to M, S be a set of maximum independent set of B, and N be a set of minimum node cover of B. We introduce two theorems about bipartite graphs, which can also be found in [17].

Theorem 6-1 [17]: M is a maximum matching of B if and only if there does not exist any augmenting path with respect to M.

Theorem 6-2 (König–Egerváry Theorem): The size of m is equal to the size of N, i.e., $|M| = |N|$.

If we remove N from B, then the remaining nodes in B form an independent set. Because N is minimum, $(H \cup V) - N$ is a maximum independent set. By definition of node cover and Theorem 6-2, for any matched pair $(v, h) \in M$, exactly one of v and h belongs to N. The following corollary follows directly.

Corollary 6-1: For any maximum independent set S of nodes in B, we have $U \subset S$ and, for any matched pair $(v, h) \in M$, either $v \in S$ or $h \in S$.

From Lemma 6-1 to Lemma 6-3, we still assume that M is an arbitrary maximum matching of chords of a simple polygon P and U is the set of unmatched chords with respect to M.

Lemma 6-1: Let $u \in U$ be an unmatched chord of a simple polygon P. If $sb(u)$ is consistent, then there is a set S of maximum nonintersecting chords such that $u \in S$ and orient(u) $\subset S$.

Proof: By Corollary 6-1, there is a set S of maximum nonintersecting chords such that $u \in S$. Let $u_i \in \text{orient}(u)$. If u_i is unmatched, then, by Corollary 6-1, $u_i \in S$. If u_i is matched with w_i and $u_i \notin S$, then, by Corollary 6-1, $w_i \in S$. For all matched $u_i \in \text{orient}(u)$ and $u_i \notin S$, we remove corresponding w_i from S and put u_i into S. After replacing, the size of S is not changed. For any chord w_j in P having the different orientation as u, there are three possibilities: both ends of w_j are not on $sb(u)$; only one of two ends of

w_j is on $sb(u)$ and both ends of w_j are on $sb(u)$. If neither end of w_j are on $sb(u)$, then w_j does not intersect with any $u_i \in \text{orient}(u)$. If only one end of w_j is on $sb(u)$, then w_j intersects with $u \in S$ and $w_j \notin S$. If both ends of w_j are on $sb(u)$, then, by our assumption that $sb(u)$ is consistent, w_j is matched with a chord $u_j \in \text{orient}(u)$ and, owing to our replacing, $w_j \notin S$. Therefore, for any $u_i \in \text{orient}(u)$, u_i does not intersect with any other chord in S and S is independent. Q.E.D.

By Lemma 6-1, if u is unmatched and $sb(u)$ is consistent, then we can put u and orient(u) into S. After that, we should not consider u and orient(u) any longer. Therefore, we execute boundary shrinking along u to eliminate $sb(u)$. Boundary shrinking has been defined in Section IV to eliminate the vertical reflex edge. In this section, boundary shrinking can be executed along a vertical chord or a horizontal chord. Figs. 10 and 11 are examples for boundary shrinkings along vertical chords. Fig. 12 is an example for the boundary shrinking along a horizontal chord. After eliminating $sb(u)$ by boundary shrinking, the chords intersecting with u will not exist in the new polygon. We use $NH(u)$ to denote the set of chords intersecting with u. Let P' be the new polygon. Let M' be the set remaining matched chords after boundary shrinking, i.e., $M' = \{v_ih_j \mid v_ih_j \in M$, both ends of v_i and both ends of h_j are on $mb(u).\}$. For any $z \in NH(u)$, z is matched with a chord w where w has the same orientation as u. If $w \in \text{orient}(u)$, then w is eliminated. If $w \notin \text{orient}(u)$, then w is in P', but does not belong to any matched pair in M' because z is deleted; that is, w is unmatched with respect to M'. Let $W = \{w \mid w$ is matched with $z \in NH(u)$ and $w \notin \text{orient}(u)\}$. Let U' be the set of unmatched chords after boundary shrinking, i.e., $U' = W \cup U - (\{u\} \cup \text{orient}(u))$.

Lemma 6-2: Let u be an unmatched chord of a simple polygon P such that $sb(u)$ is consistent. Let P', M', and U' be the polygon resulting by executing boundary shrinking along u, the remaining matched cords in P' and unmatched chords of P', respectively. Then M' is a maximum matching of chords of P' and U' is the set of unmatched cords with respect to M'.

Proof: It is obvious that M' contains only matched chords of P'. If M' is not maximum, then there are at least two chords u_i and u_j in U' such that there is an augmenting path between u_i and u_j. It is impossible that both u_i and u_j are in W because all chords in W have the same orientation. It is impossible either that both u_i and u_j are in $U - (\{u\} \cup \text{orient}(u))$ because, by Theorem 6-1, there is no augmenting path between any u_i and u_j in U. Therefore, we have $u_i \in W$ and $u_j \in U - (\{u\} \cup \text{orient}(u))$. Since u_i intersects with $NH(u)$, u_i is on an alternating path starting from u. Therefore, there is an augmenting path between u and u_j, which is a contradiction because M is a maximum matching and both u and u_j are in U. Q.E.D.

Lemma 6-3: Let $u \in U$. Assume that $sb(u)$ is consistent. Let $P' = mb(u) \cup \{u\}$ be the polygon resulting from boundary shrinking along u and S' be any set of maximum nonintersecting chords of P'. Then $S' \cup \{u\} \cup \text{orient}(u)$ is a maximal set of nonintersecting chords of P.

Proof: By the definition of boundary shrinking, no chord in $\{u\} \cup \text{orient}(u)$ intersects with any chord in P'. Therefore, $S' \cup \{u\} \cup \text{orient}(u)$ is a set of nonintersecting chords. Assume that the size of $S' \cup \{u\} \cup \text{orient}(u)$ is not maximum. By Lemma 6-1, there is a maximum set S of nonintersecting chords such that $(\{u\} \cup \text{orient}(u)) \subset S$ and $|S| > |S' \cup \{u\}\ \text{orient}(u)|$. Assume that $T = S - (\{u\} \cup \text{orient}(u))$. T is a set of nonintersecting chords of P' because no chord on $sb(u)$ is in T. We have $|T| > |S'|$. Since S' is a maximum set of nonintersecting chords of P', we have $|S'| \geqq |T|$, a contradiction. Q.E.D.

Based on Lemma 6-1 to Lemma 6-3, we know that the maximum set S of nonintersecting chords of P can be found by recursively executing the following steps.

1) Find $u \in U$ such that $sb(u)$ is consistent.
2) Put $\{u\} \cup \text{orient}(u)$ into S.
3) Execute boundary shrinking along u to have P', M', and U'.
4) Let $P = P'$, $M = M'$, and $U = U'$ and go back to step 1.

In the following, we will show that for the original polygon or for the polygon resulting from a sequence of boundary shrinkings, how to find an unmatched chord u such that $sb(u)$ is consistent. Let us consider Algorithm MAIN in Section V-5.2. For each output ω (either a matched pair or an unmatched chord) of MAIN, we give ω a number to indicate the outputting sequence of ω. This number is the *processing number* of ω, $PN(\omega)$. If ω is an unmatched chord u, then $PN(u) = PN(\omega)$. If ω is a matched pair v_ih_j, then $PN(v_i) = PN(h_j) = PN(\omega)$. The processing number of each chord is *fixed* during the whole process of finding the maximum independent set in spite of the boundary shrinkings. Let P' and M' be the polygon and the maximum matching resulting after a sequence of boundary shrinking, respectively. $\boldsymbol{sb}(u)$ of u in P' is a subset of the original $sb(u)$ of u in P. Based on $sb(u)$ defined in P', orient(u) and $NH(u)$ can be defined for u in P' accordingly. In the following, we will always use P, M, U, and S to denote the original polygon, the maximum matching, the unmatched chords with respect to M, and the maximum independent set, and use P', M', U', and S' to denote the polygon, the maximum matching, the unmatched chords with respect to M', and the maximum independent set resulting from a sequence of boundary shrinkings. In the following, we assume that $U = [u_1, u_2, \cdots, u_k]$ is ordered in the ascending order of processing numbers. The sorting sequence does not require any extra time because it is exactly the outputting sequence.

Lemma 6-4: Let $u \in U$ be any unmatched chord of the original polygon P. For any unmatched chord $u_i \in U$ whose both ends are on $sb(u)$, $PN(u_i) < PN(u)$.

Proof: If u is vertical, then, by Algorithm MATCHING, u is left-free or right-free at the time of u being processed for matching. Therefore, every vertex on $sb(u)$ should have been traced before u. If u is horizontal, by Algorithm FIND_NEIGHBOR, we know that every vertex on $sb(u)$ should have been traced before u is found to be unmatched. Therefore, for any u_i whose both ends are on $sb(u)$, $PN(u_i) < PN(u)$. Q.E.D.

Lemma 6-5: Let $U = [u_1, u_2, \cdots, u_k]$ be ordered in the ascending order of processing numbers. Then, $sb(u_1)$ is consistent.

Proof: By Lemma 6-4, we know that for any unmatched chord $u_i \in U$, if both ends of u_i are on $sb(u_1)$, then $PN(u_i) < PN(u_1)$. Since u is ordered in the ascending order of processing number, there is no unmatched chord on $sb(u_1)$. By definition, $sb(u_1)$ is consistent. Q.E.D.

By Lemma 6-1 and 6-5, we know that we can put $u_i \in U$ and orient(u_1) into S. If we execute boundary shrinking along u_1, then a new polygon P^1 and relevant M^1 and U^1 will be resulted. Let $W^1 = \{w \mid w$ be matched with $z \in NH(u_1)$ and $w \notin$ orient$(u)\}$. We have $U^1 = W^1 \cup U - (\{u_1\} \cup$ orient$(u_1))$. Let $W^1 = [w_1, w_2, \cdots, w_d]$ be an arbitrary sequence of elements in W^1. If we put W^1 at the beginning of $U - (\{u_1\} \cup$ orient$(u))$, then we have $U^1 = [w_1, w_2, \cdots, w_d, u_2, u_{2+1}, \cdots u_k]$. We shall show that $sb(w_1)$ is consistent so that we can execute boundary shrinking along w_1. However, the proof for the consistence of $sb(w_1)$ is not the same as that of $sb(u_1)$. Because, if we apply Algorithm MAIN to the new polygon P^1, M^1 may not be a possible output (in fact, we have an example to show this) and, consequently, it is not possible to prove that w_1 is the first unmatched chord as u_1. In addition, the boundary shrinking along w_1 will cause more unmatching chords w_1' and we still have to prove that $sb(w_1')$ is consistent. Lemma 6-6 and Proposition 6-1 are dedicated to prove that $sb(w_1)$ and $sb(w_1')$ are consistent.

If we execute boundary shrinking along w_1, we will have a new polygon P^2, M^2, and U^2. Let $W^2 = [w_1', \cdots, w_c']$ be the unmatched chords caused by boundary shrinking along w_1. We say that W^1 is *directly* caused by u_1 and W^2 is *indirectly* caused by u_1. For any element in $w \in W^1 < W^2$, w has the same orientation as u_1. It will also be proved later that, for any $w \in W^1 \cup W^2$, $sb(w)$ is consistent. If we execute boundary shrinking along $w \in W^1 \cup W^2$, then more unmatched chords might be caused indirectly by u_1. It is easy to see that, for any w caused directly or indirectly by u_1, w has the same orientation as u_1. If we recursively repeat boundary shrinking along the unmatched chords caused directly or indirectly by u_1, then no boundary shrinking along u_2 will be executed until all unmatched chords caused by u_1 are eliminated. If we repeat this process to eliminate unmatched chords u_1, u_2, $\cdots$, u_{i-1}, then we will have a polygon P' resulting by a sequence of boundary shrinkings along u_1 to u_{i-1} and along unmatched chords caused by u_1 to u_{i-2}. Let $U' = [w_1, w_2, \cdots, w_d, u_i, u_{i+1}, \cdots u_k]$ be the relevant unmatched cords of P'. We know that $W = [w_1, w_2, \cdots, w_d] \subset U'$ is caused (directly or indirectly) by u_{i-1} where $W = [w_1, w_2, \cdots, w_d]$ is an arbitrarily sequence of W. In the following lemma and proposition, we are going to show that, for any $w \in W$, $sb(w)$ is consistent.

Lemma 6-6: Let $W = [w_1, w_2, \cdots, w_d]$ be the set of unmatched chords caused directly or indirectly by boundary shrinking along u_{i-1}. If an unmatched chord $u_j \in [u_i, u_{i+1}, \cdots, u_k]$ which has different orientation as w_f, then no ends of u_j are on $sb(w_f)$ for any $w_f \in W$.

Proof: The proof is omitted here but is available from the authors. Q.E.D.

Proposition 6-1: Let $U' = [w_1, w_2, \cdots, w_d, u_i, u_{i+1}, \cdots u_k]$ where $W = [w_1, w_2, \cdots, w_d]$ is the set of unmatched chords caused directly or indirectly by the elimination of u_{i-1}. For any $w \in W$, $sb(w)$ is consistent.

Proof: By Lemma 6-6, there is no unmatched chord u_j on $sb(w_f)$ such that u_j has different orientation as w_f. By definition, $sb(w_f)$ is consistent. Q.E.D.

By Proposition 6-1, we know that, after executing boundary shrinking along an unmatched u_{i-1} of the original polygon such that $sb(u_{i-1})$ is consistent, the next unmatched chord whose secondary boundary is consistent can be found by arbitrarily choosing an element from W caused by u_{i-1}. After all unmatched chords caused by u_{i-1} are eliminated, the remaining unmatched chord in the polygon is $U' = [u_i, u_{i+1}, \cdots u_k]$. By Lemma 6-4, it is easy to see that $sb(u_i)$ is consistent. Therefore, we can execute boundary shrinking along u_i. Repeatedly executing boundary shrinking along the unmatched cord whose secondary bound is consistent, we can find a maximum set of nonintersecting chords. The following algorithm implements the above ideas.

Algorithm INDEPENDENT(P)

input: A simple polygon P.

output: The set S of maximum nonintersecting chords of P.

steps

1) Use Algorithm MAIN to find the maximum matching m and the set of U of unmatched chords. U is organized as a stack such that the unmatched chord with the least processing number is on the top of U;
2) Do until U is empty
   u_1 = pop(U);
   If $sb(u_1)$ have been shrinked, then repeat from 2;
   traverse $sb(u_1)$ to find orient (u) and $NH(u)$;
   put u and orient(u) into S;
   find W from $NH(u)$;
   push W into U
   boundary shrinking along u_1;
3) end.

Theorem 6-3: For a simple polygon P, the set S of maximum nonintersecting chords can be found in $O(n \log \log n)$ time.

Proof: Step 1) of INDEPENDENT(P) requires $O(n \log \log n)$ time. Steps 2) and 3) require linear time. The boundary shrinking along u_1 in step 4) is actually a three-way splitting of the finger search tree P [18]. There is no further splitting for the vertices on $sb(u_1)$. Therefore, the time required for boundary shrinkings along u_i's is bounded by $O(n)$. Totally, we need $O(n \log \log n)$ time. Q.E.D.

After the maximum set S of nonintersecting chords of P is found, we then, as described in Section II, draw these chords to partition P into a set of subpolygons such that there is no cogrid vertices in any subpolygon. The minimal partition of P is equivalent to partitioning each subpolygon into minimal rectangles. The minimal partition of each subpolygon can be found by drawing a vertical line through each concave vertex in this subpolygon as described in Section II. Therefore, we have to find the vertex-edge visible pairs for each subpolygon. After the vertex-edge visible pairs of a subpolygon are found, this subpolygon can be partitioned into rectangles in a time proportional to the number of the vertices of this subpolygon. Since it takes totally $O(n \log \log n)$ time [18] to find all the vertex-edges visible pairs for all subpolygons, the total time required for partitioning P is $O(n \log \log n)$.

VII. Conclusion

We have proposed an algorithm to solve the problem of finding the maximum matching of the bipartite graph of the intersecting chords of a simple polygon. The partition problem of simple polygon can be solved in $O(n \log \log n)$ by applying our results. This time bound is equal to the time required for triangulating a nonrectilinear simple polygon. If the simple rectilinear polygon can be triangulated in linear time, then our algorithm will be an optimal one. However, our algorithm is optimal for partitioning convex and horizontally convex rectilinear polygons.

Acknowledgment

The authors wish to express their thanks to the referee whose valuable comments improved the presentation and simplified the proofs of this paper.

References

[1] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *The Design and Analysis of Computer Algorithms*. Reading, MA: Addison-Wesley, 1974.
[2] S. Chaiken, D. J. Kleitman, M. Saks, and J. Shearer, "Covering regions by rectangles," *SIAM J. Algebraic Discrete Methods*, vol. 2, pp. 394–410, 1981.
[3] L. Ferrari, P. V. Sankar, and J. Sklansky, "Minimal rectangular partition of digitized blobs," *Comput. Vision, Graph. Image Proc.*, vol. 28, pp. 58–71, 1984.
[4] D. S. Franzblau and D. J. Kleitman, "An algorithm for covering polygons with rectangles," *Inform. Contr.*, vol. 63, pp. 164–189, 1984.
[5] H. N. Gabow and R. E. Tarjan, "A linear-time algorithm for a special case of disjoint set union," *J. Comput. Syst. Sci.*, vol. 30, pp. 209–221, 1985.
[6] F. Glover, "Maximum matching in convex bipartite graph," *Naval Res. Logist. Quart.*, vol. 14, pp. 313–316, 1967.
[7] K. D. Gourley and D. M. Green, "A polygon-to-rectangle conversion algorithm," *IEEE Comput. Graph. Appl.*, vol. 3, pp. 31–36, 1983.
[8] A. Hegedüs, "Algorithms for covering polygons by rectangles," *Computer Aided Design*, vol. 14, pp. 257–260, 1982.
[8] E. Hopcroft and R. M. Karp, "An $n^{5/2}$ algorithm for maximum matchings in bipartite graph," *SIAM J. Comput.*, vol. 2, pp. 225–231, 1973.
[10] H. Imai and T. Asano, "Efficient algorithms for geometric graph search problems," *SIAM J. Comput.*, vol. 15, pp. 478–494, 1986.
[11] W. Lipski, Jr., E. Lodi, F. Luccio, C. Muganai, and L. Pagli, "On two dimensional data organization II: *Fundamenta Informaticae*, vol. 2, pp. 245–260, 1979.
[12] W. Lipski, Jr., and F. P. Preparata, "Efficient algorithms for finding maximum matchings in convex bipartite graphs and related problems," *Acta Informat.*, vol. 15, pp. 329–346, 1981.
[13] K. Mehlhorn, *Data Structures and Algorithms 1: Sorting and Searching*. Berlin, Germany: Springer-Verlag, 1984.
[14] S. Nahar, and S. Sahni, "Fast algorithm for polygon decomposition," *IEEE Trans. Computer-Aided Design*, vol. 7, pp. 473–483, 1988.
[15] T. Ohtsuki, "Minimum dissection of rectilinear regions," in *Proc. IEEE Symp. on Circuits and Systems*, 1982, pp. 1210–1213.
[16] T. Ohtsuki, M. Sato, M. Tachibana, and S. Torii, "Minimum partitioning of rectilinear regions," *Trans. Inform. Proc. Soc. Japan.*, 1983.
[17] C. H. Papadimitriou, and K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity*. Englewood Cliffs: Prentice-Hall, 1982.
[18] R. E. Tarjan, and C. J. Van Wyk, "An $O(n \log \log n)$ time algorithm for triangulating simple polygons," *SIAM J. Comput.*, vol. 17, pp. 143–178, 1988.

*

W. T. Liou received the B.E. degree in 1975 from the Department of Naval Architecture and Marine Engineering, National Cheng Kung University, the M.A. degree in 1985 from the Institute of Industrial Engineering, National Tsing Hua University, and the Ph.D. degree in 1989 from the Institute of Computer Science, National Tsing Hua University.

Currently, he is with the Department of Management Information Systems, National ChengChi University. His research interests include algorithm design, neural net computing, and pattern recognition.

*

Jimmy Jiann-Mean Tan received the B.S. and M.S. degrees in mathematics from the National Taiwan University in 1970 and 1973, respectively, and the Ph.D. degree from Carleton University, Ottawa, Canada, in 1981.

He has been on the faculty of the Department of Computer and Information Science, National Chiao Tung University, since 1983. His research interests include design and analysis of algorithms, combinatorial optimization, and graph theory.

*

R. C. T. Lee (M'75-SM'86-F'89) received the B.S. degree in electrical engineering from the National Tsing Hua University and the M.S. and Ph.D. degrees in electrical engineering and computer science from the University of California, Berkeley, in 1961, 1963, and 1967, respectively.

He has worked for the National Institute of Health and the Naval Research Laboratory. In 1975, he joined the National Tsing Hua University, where he has held the positions of Department Chairman for Applied Mathematics, Computer Science, and Electrical Engineering, and Dean of Engineering. Currently, he is the Dean of Academic Studies. He has published fifty papers on computer science and is the co-author of *Symbolic Logic* and *Mechanical Theorem Proving* (Academic Press).

Dr. Lee is an editor for the *International Journal of Pattern Recognition, Machine Intelligence, Annuals of Mathematics* and *Artificial Intelligence* and the IEEE Transactions on Knowledge and Data Engineering. He is also a reviewer for *Mathematical Reviews*.