Minimum Partitioning Simple Rectilinear Polygons in O(nloglogn)–Time

W.T. Liou[*], J.J.M. Tan[**], and R.C.T. Lee[***]

[*] Institute of Computer Science, National Tsing Hua University, Hsinchu, Taiwan, R.O.C.
[**] Institute of Information Science, National Chiao Tung University, Hsinchu, Taiwan, R.O.C.
[***] National Tsing Hua University, Hsinchu, Taiwan, and the Academia Sinica, Taipei, Taiwan, R.O.C.

1 Introduction

The minimum rectangular partition problem for a simple rectilinear polygon is to partition the interior of a simple rectilinear polygon into minimum number of rectangles. This problem is related to VLSI mask generation. A VLSI mask is usually a piece of glass with a figure engraved on it. The engraved figure can be viewed as a rectilinear polygon on the digitized plane [OHTS82]. In order to engrave the figure on the VLSI mask, a pattern generator is often used. A traditional pattern generator has a rectangular opening for exposure, which exposes rectangles onto the mask. Therefore, the engraved figure has to be decomposed into rectangles such that the pattern generator can expose each of these rectangles. The number of rectangles will determine the time required for mask generation. Therefore, decomposing a rectilinear polygon into minimum number of rectangles is an important problem for optimal automated VLSI mask fabrication. The decomposition can be classified into two types depending on the resulted rectangles. If the resulted rectangles can not overlap with each other, then the decomposition is a partition. If the resulted rectangles overlap with each other, then the decomposition is a cover. Both partitioning approach and covering approach for VLSI mask generation have been discussed in previous researches such as [LIPS79, OHTS82, GOUR83, FERR84, IMAI86, NAHA88] for partitioning problems and [CHAI81, HEGE82, FRAN84] for covering problems. In this paper, we shall only consider the partitioning problem for simple rectilinear polygons. The time complexity of our approach is O(nloglogn). The partition problem for convex rectilinear polygons or vertically (horizontally) convex polygons can be solved in linear time which is optimal. As for a rectilinear polygon with holes, we prove that O(nlogn) is a lower bound, though, as far as we know, there is no algorithm achieve this bound.

2 Previous Results

The minimum rectangular partition problem has been studied in [LIPS79, OHTS82, OHTS83, FERR84, IMAI86]. Some of their results are discussed below which 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 non–overlapping 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 270^0 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 cogrid if they are cohorizontal $(y_1=y_2)$ or covertical $(x_1=x_2)$. 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 chords. Ferrari, Sankar and Sklansky [FERR84] 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 Figure 2–1(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 Figure 2–1(b). Note that there might be other approaches to solve the minimal rectangular partition problem. However, this approach which is based upon finding a largest set of nonintersecting chords will definitely lead to a minimal solution.

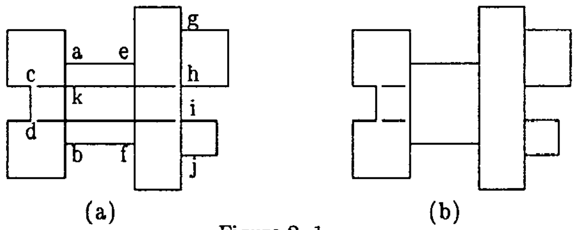

Figure 2–1

In [FERR84], 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_ih_j in E corresponds to the intersection of chords i and j.
3 Find a maximum matching [PAPA82] M of B.
4 Find a maximum independent set [PAPA82] 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 cogrid 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 [OHTS82] and [FERR84], Hopcroft and Karp's algorithm [HOPC73] 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 [IMAI86] 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. But 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 [IMAI86], 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.

3 Maximum Matching of a Special Bipartite Graph

In this section, we shall introduce a theorem which was initially discussed by Glover [GLOV67] and, then, generalized by Lipski and Preparata [LIPS81]. 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_i in V and $V(h_g)$ to denote the neighbors of a vertex h_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).

We define that $H_g(v_i)=\{h_g | h_g\in H(v_i)$ and $H(v_i)\subset H(v_j)$, for all $v_j\in V(h_g)\}$. By Theorem 3–1, for any $h_g\in H_g(v_i)$, there is a maximum matching containing (v_i,h_g). 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 $h_g\in H_g(v_i)$ 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 as shown in Figure 3–1. 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 $H(v_i)$.

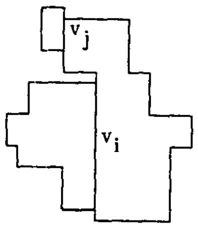

Figure 3–1

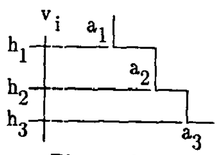

Figure 3–2

Lemma 3–1 Let v_i be a left–free vertical chord. For any vertical chord v_j, if $x(v_j)\leq x(v_i)$, 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 more complicated. Let v_i be a vertical chord of a simple polygon. Let $h_1,h_2,\ldots,h_k$ be the horizontal chords of $H(v_i)$ sorted in the descending order of y–position. Let $a_1,a_2,\ldots,a_k$ be the right ends of $h_1,h_2,\ldots,h_k$, 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 $[a_1,a_2,\ldots,a_k]$ because P is simply connected. (See Figure 3–2.)

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 vertices 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 is 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 chords in $H(v_i)$.

Proof Let $A=[a_1, a_2, \ldots,a_k]$ be the set of right ends of $H(v_i)$ 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 this 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 Figure 3–3.) 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 $y(p_2)>y(p_2')$ as shown in Figure 3–3(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 Figure 3–3(b).

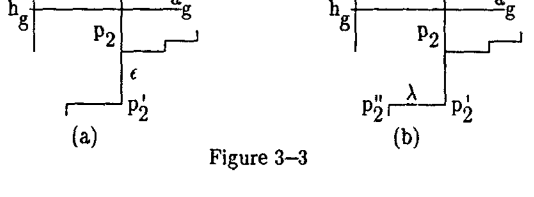

Figure 3–3

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 of p_2. We have $x(p_2)=x(p_2')$. (1) If $p_2'=a_f$, then $\overline{p_2a_f}$ 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 $\overline{p_2p_2'}$ is a vertical reflex edge on ΔP, a contradiction. If p_2' is a convex 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 Lemma 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) vertical 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_ih_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 Lemma 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 $v_ih_g \in M$. Q.E.D.

4 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 illustrate will be later extended to simple polygons.

Consider a simple polygon P. 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. The polygon in Figure 4–1(a) is a horizontally convex polygon.

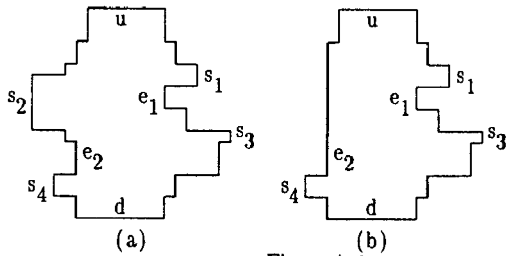

Figure 4–1

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 $x(s_i)<x(v_k)\leq x(e_i)$. For e_i and s_i on the right chain, a vertical chord is located between e_i and s_i if $x(s_j)>x(v_k)\geq x(e_i)$. The eliminating of the boundary right (left) to the extension of a vertical edge is called the boundary shrinking along this extension. The boundary shrinking along the extension of e_2 is shown in Figure 4–1(b).

In general, for a given horizontally convex polygon, 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 leftmost 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 the 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 chords located between $s_i$ and $e_i$, then
   execute boundary shrinking along $e_i$; otherwise, find the
   vertical chord $v_i$ located between $s_i$ 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 $v_ih_g$ 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=MUM';
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_1h_g. After v_1 and h_g are matched, we move the x–position of the upper end of v_1 a small distance (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 y–direction such that h_g will not exist and no other horizontal chords will be produced. The resulting polygon is still a horizontally 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 $M^2 \cup \{v_1h_g\}$. 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.

In the proof of Theorem 4–1, we adjust the positions of ends of chords to eliminate chords after matching. This adjustment is only for our induction proof. In practical processing, we do not adjust boundary except boundary shrinking.

Having shown how we can handle the case of horizontally convex polygons, we shall now show that we can handle the simple polygons.

5 Maximum Matching of a Simple Polygon

This section includes two subsections. In subsection 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 subsection 5.2, we will explain our algorithm in detail and analyze the time required for executing our algorithm.

5.1 Basic Ideas

A simple polygon can be partitioned into horizontally convex sub–polygons by drawing horizontal extensions through each horizontal reflex edge. For example, the simple polygon in Figure 5–1(a) is partitioned into five horizontally convex sub–polygons. After this, we construct a tree of these horizontally convex sub–polygons by the following rule: Two horizontally convex sub–polygons are connected if and only if they share one horizontal extension through some horizontal reflex edge. For the case shown in Figure 5–1(a), the tree is shown in Figure 5–1(b).

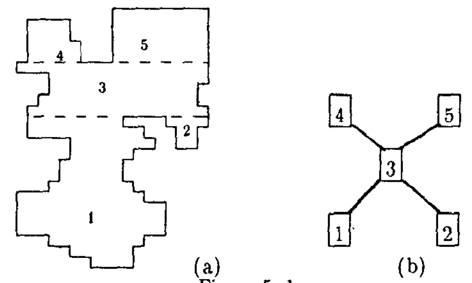

Figure 5–1

After the tree is constructed, we arbitrarily choose a node as the root and, then, the horizontally convex sub–polygons are processed according to the postorder [AHO74] sequence. For the tree in Figure 5–1(b), if we assign node 5 as the root, then a postorder sequence is 1,2,4,3,5. For a node n_i of the resulting tree, 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. We define vertically visible for a concave vertex ν and a horizontal edge ϵ. ν is vertically visible to ϵ if we draw a vertical extension through ν and ϵ is the first hit edge. If ν is vertically visible to the master bound, then it is possible that there exists a vertical chord through ν and this chord can not be found in n_i. We process each sub–polygon in the postorder sequence using the method described in Section 4 with some modifications shown below.

1 When we process node n_i, we do not process the vertices which are vertically visible to the master bound of n_i.
2 Vertical edges at each end of the master bound are viewed as vertical reflex edges. We call them as virtual vertical reflex edges.

Consider sub–polygon 1. We process sub–polygon 1 as described in Section 4 with the above modifications. After all vertical reflex edges (including the virtual vertical reflex edges) are eliminated, the resulting sub–polygon is shown in Figure 5–2(a). Note that the remaining vertices in the sub–polygon are all vertically visible to the master bound of sub–polygon 1. We then merge sub–polygon 1 to its parent, sub–polygon 3, as shown in Figure 5–2(b).

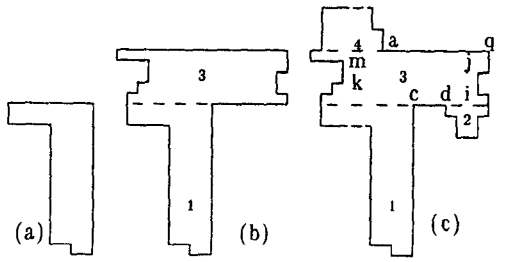

Figure 5–2

Note that, after merging with child nodes, the internal nodes may not be a horizontally convex sub–polygon. For example, the resulting sub–polygon 3 after merging with sub–polygons 1,2, and 4 is not a horizontally convex sub–polygons because the horizontal reflex edge **cd** exists as shown in Figure 5–2(c). But 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 or on 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 Figure 5–2(c), the vertical reflex edges **ji** and **mk** are on the boundary between the master bound and the slave bound. We still can find the vertical reflex edges and compare their heights. 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 sub–polygons and
    construct a tree T of horizontally convex sub–polygons;
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 postorder 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.*/
        $M=M \cup M_i$;
        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, though 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 sub–polygon 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, Algorithm 2 will be modified in subsection 5.2 to find a maximum matching in O(nloglogn) 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 6 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 sub–polygons.
    Construct a tree $T_h$ of horizontally convex sub–polygons;
    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 postorder;
4   For each visited node $n_k$ of $T_h$, do
        Find the maximum matching of chords in $n_k$;
        If $n_k$ in 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 ν of P, there are two data items about ν. One is the position of ν on the plane and the other is that ν is the ith input element of P. For the ith vertex ν, we can locate ν in R(P) in constant time. In order to partition P into horizontally convex sub–polygons, we have to draw horizontal extensions through horizontal reflex edges to hit the nearest boundary. Tarjan and Van Wyk [TARJ88] 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 ν, if the vertex–edge visible pair of ν is known, the hit point of ν can be computed in constant time. In [TARJ88], Tarjan and Van Wyk proposed an O(nloglogn) 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 ν of P, we can decide the hit point of ν in constant time.

In addition to R(P), P is also stored in a finger search tree [TARJ88] which can be constructed in linear time. Using a concave vertex ν and its hit point hit(ν), the finger search tree of P is partitioned into two sub–trees (sub–polygons): one contains the vertices from ν to hit(ν) and the other contains rest vertices. This is a three–way splitting [TARJ88]. Each three–way splitting will reduce one end of horizontal reflex edges. For each sub–polygon, 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 ν to hit(ν) (hit(ν) is exclusive), it takes $O(1+\log(\min\{i,n-i\}+1))$ amortized time [TARJ88] to split this finger search tree into two sub–trees containing i and n–i vertices, respectively. (Note that, though the hit point will be a vertex in each sub–polygon, we do not include these two new vertices in the sub–trees. 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.

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

Solving this recurrence, we have T(n)=O(n) [MEHL84]. Therefore, the total time required for partitioning P into horizontally convex sub–polygons is O(nloglogn).

The tree T_h of horizontally convex sub–polygons is constructed as follows. Initially, there is only one node in T_h corresponding to the original finger search tree. Whenever the finger search tree or a sub–finger 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 sub–polygons. After constructing T_h, arbitrarily assign a node of T_h as the root.

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 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 sub–polygons. Tree T_v is constructed in a similar way as T_h. 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. Finding new positions of horizontal chords can be executed on T_v using the <u>static tree set union</u> algorithm of Gabow and Tarjan [GABO85]. In [GABO85], 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 [GABO85]. 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 connecting 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 [GABO85] 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 bounded by O(n). Therefore, O(m+k) is bounded by O(n).

Step 4 The ideas for finding maximum matching of a node of T_h have been explained in subsection 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 <u>upper chain</u> and the <u>lower chain</u>. 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 be <u>unmatched</u>. 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 <u>unmatched</u>. <u>The unmatched chords are put into a set U</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 of the right end of h is traced by FIND_NEIGHBOR, h is the tail (head) of the list $H(v_i)$.

<u>Algorithm FIND_NEIGHBOR(v_{i-1},v_i)</u>

```
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.*/
    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 $H(v_{i-1})$ have 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 <u>traced by FIND_NEIGHBOR before</u> 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 $\overline{p_i hit(p_i)}$ slices the polygon into two pieces. Then, either the vertices on the left–piece or the vertices on the right–piece of $\overline{p_i hit(p_i)}$ 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 subsection 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 $v_i h_g$ 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 $U=U\cup H(v_i)$;
end;
```

In MAIN, the time required for partitioning P into horizontally convex sub–polygons is O(nloglogn) 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(nloglogn).

Theorem 5–2 The total time required for Algorithm MAIN to find the maximum matching of chords of a simple polygon P is O(nloglogn) 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 sub–polygons. 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.

6 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 5.

For any horizontal or vertical chord u ot P, u slices the boundary of P into two pieces. One piece contains the master bound and this piece is the main boundary of u, denoted as mb(u). The other piece is the secondary boundary of u, denoted as sb(u). Two chords have the same orientation if they are both vertical or both horizontal. orient(u) is a set of chords such that, for any $t\in$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., $u'\in$orient(u). Boundary shrinking has been defined in Section 4 to eliminate the vertical reflex edge. In this section, boundary shrinking can be executed along a vertical chord or 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. If a chord is matched with an element in NH(u) but both ends of this chord are not on sb(u), then this chord will be an unmatched chord after boundary shrinking of sb(u). Let W be the set of unmatched chords which are caused by eliminating sb(u). We propose the following algorithm to find the maximum nonintersecting chords.

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 U of unmatched chords. U is organized as a
     stack and the unmatched chords in U are sorted such that
     the first found unmatched chord is on the top of U and the
     last found unmatched chord is at the bottom 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.
```

In order to prove that the above algorithm 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 a 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 MAIM.

(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 $S=\{u_1\}\cup$orinet(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–{chords on sb(u_1)} is a maximal matching of the new polygon. Also in Lemma 6–2, we prove that $W_1\cup U$–($\{u_1\}\cup$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 we order elements of U–($\{u_1\}\cup$orient(u_1)) according to their outputting sequence and put W_1 at the beginning of U–($\{u_1\}\cup$orient(u_1)) to have a new sequence $U'=[w_1,\ldots w_k,u_2,\ldots,u_n]$, 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'.

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.

Theorem 6–1 [PAPA82] 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 (Kõnig–Egervàry Th.) 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∪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⊂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)⊂S.

Proof By Corollary 6–1, there is a set S of maximum nonintersecting chords such that $u\in S$. Let $u_i\in$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 $u_i\in$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 ends of w_j are on sb(u), then w_j does not intersect with any $u_i\in$orient(u). If only one end of w_j is on sb(u), then w_j intersect 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 $u_j\in$orient(u) and, owing to our replacing, $w_j\notin S$. Therefore, for any $u_i\in$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). After eliminating sb(u) by boundary shrinking, the chords intersecting with u will not exist in the new polygon. Let P' be the new polygon. Let M' be the set of remaining matched chords after boundary shrinking, i.e., $M'=\{v_ih_j|v_ih_j\in M$, 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 $w\in$orient(u), then w is eliminated. If $w\notin$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 $W=\{w|w$ is matched with $z\in$NH(u) and $w\notin$orient(u)$\}$. Let U' be the set of unmatched chords after boundary shrinking, i.e., $U'=W\cup U-(\{u\}\cup$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 chords 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 chords with respect to M'.

Proof It is obvious that M' contains only matched chords of P'. If M' is not maximum, then there is 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}∪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 $u_j\in U-(\{u\}\cup$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)∪{u} be the polygon resulting from boundary shrinking along u and S' be any set of maximum nonintersecting chords of P'. Then S'∪{u}∪orient(u) is a maximal set of non–intersecting chords of P.

Proof By the definition of boundary shrinking, no chord in {u}∪orient(u) intersects with any chord in P'. Therefore, S'∪{u}∪orient(u) is a set of nonintersecting chords. Assume that the size of S'∪{u}∪orient(u) is not maximum. By Lemma 6–1, there is a maximum set S of nonintersecting chords such that ({u}∪orient(u))⊂S and $|S|>|S'\cup\{u\}\cup$orient(u)$|$. Assume that T=S–({u}∪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 $|S'|\geq|T|$, a contradiction. **Q.E.D.**

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 subsection 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(ω). If ω is an unmatched chord u, then PN(u)=PN(ω). If ω is a matched pair v_ih_j, then PN(v_i)=PN(h_j)= PN(ω). 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. 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 $U=[u_1,u_2,...,u_k]$ is ordered in the ascending order of processing numbers. This 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 $U=[u_1,u_2,...,u_k]$ 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 $u_1\in U$ 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 $W^1=\{w|w$ is 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 $W^1=[w_1,w_2,...,w_d]$ be an arbitrary sequence of elements in W^1. If we put W^1 at the beginning of U–({u_1}∪orient(u_1)), then we have $U^1=[w_1,w_2,...,w_d,u_2,u_{2+1},...u_k]$. We shall show that sb($w_1$) is consistent so that we can execute boundary shrinking along w_1.In addition, the boundary shrinking along w_1 will cause more unmatched 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 $W^2=[w'_1,...,w'_c]$ 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 $w \in W^1 \cup W^2$, 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 chord 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 $u_1,u_2,...,u_{i-1}$, 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 $U'=[w_1,w_2,...,w_d,u_i,u_{i+1},...u_k]$ be the relevant unmatched chords of P'. We know that $W=[w_1,w_2,...,w_d] \subset U'$ is caused (directly or indirectly) by u_{i-1} where $W=[w_1,w_2,...,w_d]$ 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 $W=[w_1,w_2,...,w_d]$ be the set of unmatched chords caused directly or indirectly by boundary shrinking along u_{i-1}. If an unmatched chord $u_j \in [u_i,u_{i+1},...,u_k]$ 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 Omitted in this conference paper.
Q.E.D.

Proposition 6–1 Let $U'=[w_1,w_2,...,w_d,u_i,u_{i+1},...u_k]$ where $W=[w_1,w_2,...,w_d]$ 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.**

Theorem 6–3 Algorithm INDEPENDENT correctly finds the maximum nonintersecting chords.

Proof By Proposition 6–1, we know that, after executing boundary shrinking along an unmatched chord 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 chord caused by u_{i-1} are eliminated, the remaining unmatched chord in the polygon is $U'=[u_i,u_{i+1},...u_k]$. 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 chord whose secondary boundary is consistent, we can find a maximum set of nonintersecting chords. Q.E.D.

Step 1 of INDEPENDENT(P) requires O(nloglogn) 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 of P [TARJ88]. There is no further splitting for the vertices on $sb(u_1)$. Therefore, the time required for boundary shrinkings along u_1's is bounded by O(n). Totally, we need O(nloglogn) time. After the maximum set S of nonintersecting chords of P is found, we then, as described in Section 2, draw these chords to partition P into a set of sub–polygons such that there is no cogrid vertices in any sub–polygon. The minimal partition of P is equivalent to partitioning each sub–polygon into minimal rectangles. The minimal partition of each sub–polygon can be found by drawing a vertical line through each concave vertex in this sub–polygon as described in Section 2.

Section 7 Lower Bounds of the Partition Problem

In this section, we discuss the lower bound problem for partitioning a polygon with holes. We prove that O(nlogn) operations are required to solve the partition problem for polygons with holes by reducing the sorting problem to this problem. The reducing strategy is inspired by [ASAN86].

Consider a set $X=\{x_1,x_2,...,x_n\}$ of n distinct positive integers. Let m and M be the smallest and the biggest integers in X, respectively. A polygon P is constructed as follows. First, a rectangle $R=[(m,0),(M+\epsilon,3)]$ is constructed where ϵ is a small fraction, e.g., 0.1, (m,0) is the lower–left corner, and $(M+\epsilon,3)$ is the upper–right corner of R. We then draw two vertical reflex edges λ_1 and λ_2 along the vertical sides at m and $M+\epsilon$ as shown in Figure 7–1.

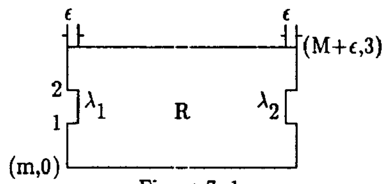

Figure 7–1

For each $x_i \in X-\{m,M\}$, a rectangular hole $h_i=[(x_i,1),(x_i+\epsilon,2)]$ is added inside R. (See Figure 7–2(a).) There are totally n–2 holes in R. The polygon P is equal to the union of R and h_i.

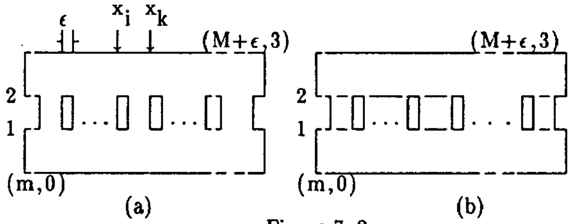

Figure 7–2

The minimum partition of this polygon is shown in Figure 7–2(b). In Figure 7–2(b), there are n+1 rectangles in the partition. Among these n+1 rectangles, there are n–1 rectangles whose upper side is at height 2, lower side is at height 1, left side is at $x_i+\epsilon$ and right side is at x_k. It is obvious that x_k is the integer succeeding x_i. We can sort integers in X using these n–1 rectangles as follows. We construct an array NEXT(2,n–1). For each rectangle $[(x_i+\epsilon,1),(x_k,2)]$, we set $NEXT(1,i)=x_i$ and $NEXT(2,i)=k$ (see Figure 7–3). That is, we store x_i at the ith element and set a pointer from the ith element to the kth element. Tracing NEXT in linear time, we can find the successor x_k of each x_i.

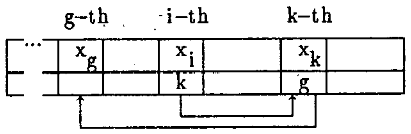

Figure 7–3 Array NEXT

Lemma 7–1 O(nlogn) is a lower bound for the partitioning problem for polygons with holes.

8 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(nloglogn) by applying our results. This time bound is equal to the time required for triangulating a non–rectilinear simple polygon. We have also shown that the lower bound for the minimum partition problem for polygons with holes is O(nlogn) which is the same as triangulating a non–rectilinear polygon with holes [ASAN86].

Though triangulation problem is not a minimization problem, it seems interesting to note the similarities between the time complexities of the minimum rectangular partition problem and the triangulation problem.

References

[AHO74] A.V. Aho, J.E. Hopcroft and J.D. Ullman, The Design and Analysis of Computer Algorithms, Addison–Wesley, Reading, Mass., 1974.

[ASAN86] T. Asano, T. Asano, and R. Y. Pinter, 'Polygon Triangulation : Efficiency and Minimality', Journal of Algorithms 7(1986), 221–231.

[CHAI81] S. Chaiken, D.J. Kleitman, M. Saks, and J. Shearer, 'Covering Regions by Rectangles', SIAM Journal on Algebraic and Discrete Methods 2(1981), 394–410.

[FERR84] L. Ferrari, P.V. Sankar, and J. Sklansky, 'Minimal Rectangular Partition of Digitized Blobs', Computer Vision, Graphics, and Image Processing 28(1984), 58–71.

[FRAN84] D.S. Franzblau, and D.J. Kleitman, 'An Algorithm for Covering Polygons with Rectangles', Information and Control 63(1984), 164–189.

[GABO85] H.N. Gabow, and R.E. Tarjan, 'A Linear–Time Algorithm for a Special Case of Disjoint Set Union', Journal of Computer and System Sciences 30(1985), 209–221.

[GLOV67] F. Glover, 'Maximum Matching in Convex Bipartite Graph', Naval Res. Logist. Quart. 14(1967), 313–316.

[GOUR83] K.D. Gourley, and D.M. Green, 'A Polygon–to–Rectangle Conversion Algorithm', IEEE Computer Graphics and Applications 3(1983), 31–36.

[HEGE82] A. Hegedüs, 'Algorithms for Covering Polygons by Rectangles', Computer Aided Design 14(1982), 257–260.

[HOPC73] E. Hopcroft, and R.M. Karp, 'An $n^{5/2}$ Algorithm for Maximum Matchings in Bipartite Graph', SIAM Journal on Computing 2(1973), 225–231.

[IMAI86] H. Imai, and T. Asano, 'Efficient Algorithms for Geometric Graph Search Problems', SIAM Journal on Computing 15(1986), 478–494.

[LIPS79] W. Lispki, Jr., E. Lodi, F. Luccio, C. Muganai, and L. Pagli, 'On Two Dimensional Data Organization II,' Fundamenta Informaticae 2(1979), 245–260.

[LIPS81] W. Lipski, Jr., and F.P. Preparata, 'Efficient Algorithms for Finding Maximum Matchings in Convex Bipartite Graphs and Related Problems', Acta Informatica 15(1981), 329–346.

[MEHL84] K. Mehlhorn, Data Structures and Algorithms 1: Sorting and Searching, Springer–Verlag, Berlin, 1984.

[NAHA88] S. Nahar, and S. Sahni, 'Fast Algorithm for Polygon Decomposition', IEEE Transactions on Computer–Aided Design of Integrated Circuits and Systems 7(1988), 473–483.

[OHTS82] T. Ohtsuki, 'Minimum Dissection of Rectilinear Regions', Proceedings of IEEE Symposium on Circuits and Systems, 1982, pp.1210–1213.

[OHTS83] T. Ohtsuki, M. Sato, M. Tachibana, and S. Torii, 'Minimum Partitioning of Rectilinear Regions', Transaction of Information Processing Society of Japan, 1983.

[PAPA82] C.H. Papadimitriou, and K. Steiglitz, Combinatorial Optimization : Algorithms and Complexity, Printice–Hall, Inc., New Jersey, 1982.

[TARJ88] R.E. Tarjan, and C.J. Van Wyk, 'An O(nloglogn) time Algorithm for Triangulating Simple Polygons', SIAM Journal on Computing 17(1988), 143–178.