Algorithmica (1996) 16: 339–357

Algorithmica

Shortest Noncrossing Paths in Plane Graphs

Jun-ya Takahashi,[1] Hitoshi Suzuki,[2] and Takao Nishizeki[2]

Abstract. Let G be an undirected plane graph with nonnegative edge length, and let k terminal pairs lie on two specified face boundaries. This paper presents an algorithm for finding k "noncrossing paths" in G, each connecting a terminal pair, and whose total length is minimum. Noncrossing paths may share common vertices or edges but do not cross each other in the plane. The algorithm runs in time $O(n \log n)$ where n is the number of vertices in G and k is an arbitrary integer.

Key Words. Noncrossing paths, Shortest path, Plane graphs, Single-layer routing, VLSI.

1. Introduction. The shortest disjoint path problem, that is, to find k vertex-disjoint paths with minimum total length, each connecting a specified terminal pair, in a plane graph G has many practical applications such as VLSI layout design. The problem is NP-complete [L], [KL], and so it is very unlikely that there exists a polynomial-time algorithm for its solution. However, if two or more wires may pass through a single routing region [DAK], then the problem can be reduced to the shortest "noncrossing" path problem. Here "noncrossing" paths may share common vertices or edges but do not cross each other in the plane. The shortest noncrossing path problem is expected to be solvable in polynomial time at least for a restricted case, for example, a case where either the number k of paths or the number of face boundaries on which all the terminals are located is bounded. Indeed an $O(n \log n)$ algorithm has been obtained for the special case of $k = 2$, where n is the number of vertices in G [LSYW].

In this paper we present an $O(n \log n)$ algorithm to find shortest noncrossing paths in a plane graph for the case when all the terminals of k pairs are located on two specified face boundaries. We assume that k is an arbitrary integer. For the same case, Suzuki *et al.* [SAN1], [SAN2] obtained an $O(n \log n)$ algorithm for finding *vertex-disjoint* paths, but the total length of the paths found by their algorithm is not minimum. Our algorithm can be applied to a single-layer routing problem which appears in the final stage of VLSI layout design, where each wire connects a pad on the boundary of the chip and a pin on the boundary of a block (see Figure 1). Furthermore, we show that a similar algorithm can find noncrossing paths that are optimal with respect to any objective nondecreasing function in the length of each path.

The rest of the paper is organized as follows: In Section 2 we give a formal description of the problem and define terms. In Section 3 we present an algorithm to find shortest noncrossing paths in a plane graph G for the case where all terminals lie on a single face

[1] Department of Computer Science, Faculty of Engineering, Iwate University, Morioka 020, Japan.
[2] Graduate School of Information Sciences, Tohoku University, Sendai 980-77, Japan.

Received April 12, 1994; revised October 23, 1994. Communicated by K. Melhorn.

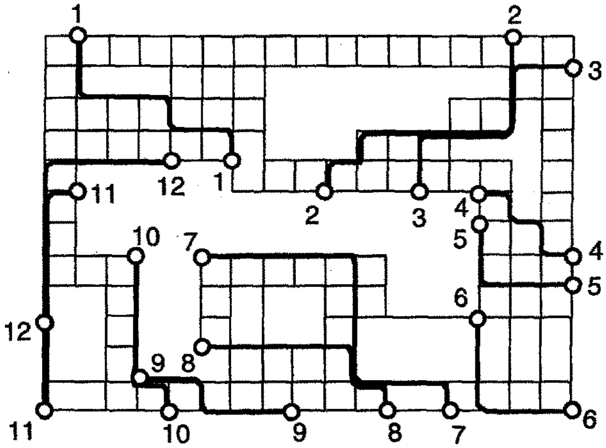


Fig. 1. Noncrossing paths in a grid graph.

boundary. A naive algorithm for this case takes time $O(kn \log n)$, but our algorithm takes time $O(n \log n)$. The main idea behind the algorithm is a divide-and-conquer technique based on the "genealogy tree" of terminal pairs (see Figure 6). In Section 4 we present an $O(n \log n)$ algorithm to find shortest noncrossing paths for the case where all terminals lie on two face boundaries B_1 and B_2. There are two main ideas behind the algorithm. The first idea is to notice that there exists a solution to the problem which contains one of three specified paths connecting a terminal on B_1 and a terminal on B_2, i.e., either a shortest such path or one of two certain induced paths. The second idea is to reduce an instance of the problem to three instances of the former problem by "slitting" the graph along these three paths such that all terminals lie on a single face boundary in the resulting graphs. In Section 5 we present an algorithm to find optimal noncrossing paths. Finally, we conclude in Section 6 with some general comments. A preliminary version of this paper was presented in [TSN].

2. Preliminaries. In this section we give a formal description of the noncrossing path problem and define terms. We denote by $G = (V, E)$ a graph consisting of vertex set V and edge set E. We denote by $V(G)$ and $E(G)$ the vertex and edge sets of G, respectively. Assume that G is an undirected plane graph and that every edge in G has a nonnegative edge length. Furthermore we assume that G is embedded in the plane $\mathbb{R}^2$. The image of G in $\mathbb{R}^2$ is denoted by $Image(G) \subset \mathbb{R}^2$. A *face* of G is a connected component of $\mathbb{R}^2 - Image(G)$. The *boundary* of a face is the maximal subgraph of G whose image is included in the closure of the face. For two subgraphs $H_1 = (V_1, E_1)$ and $H_2 = (V_2, E_2)$, we define $H_1 + H_2 = (V_1 \cup V_2, E_1 \cup E_2)$. A pair of vertices s_i and t_i which we wish to connect by a path is called a *terminal pair* (s_i, t_i). Let S be the set of terminal pairs, and let k be its cardinality. In this paper we assume that k is an arbitrary integer. Let all the terminals be located on boundaries B_1 and B_2 of two specified faces f_1 and f_2. We can assume without loss of generality that G is 2-connected, $V(B_1) \cap V(B_2) = \emptyset$, and all

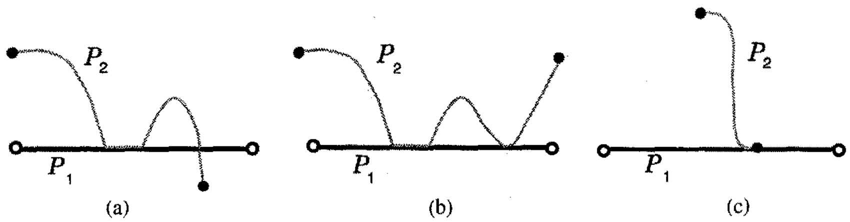


Fig. 2. Crossing paths (a) and noncrossing paths (b), (c).

terminals are distinct from each other, because one may replace a vertex in G with its two copies and an edge joining them and having length 0 if necessary.

For paths P_1 and P_2 depicted in Figure 2(a), $Image(P_1)$ and $Image(P_2)$ cross each other on the plane. On the other hand $Image(P_1)$ and $Image(P_2)$ do not cross each other in Figures 2(b) and (c). Let $P_1, P_2, \ldots, P_k$ be paths connecting the k terminal pairs. Let G^+ be a plane graph obtained from G as follows: add a new vertex v_{f_1} in face f_1, and join v_{f_1} to each terminal on B_1; similarly, add a new vertex v_{f_2} in face f_2, and join v_{f_2} to each terminal on B_2. Let P_i', $1 \le i \le k$, be a path (or a cycle) in G^+ obtained from P_i by adding two new edges: one joins s_i to v_{f_1} if s_i is on B_1, otherwise to v_{f_2}; and the other joins t_i to v_{f_1} if t_i is on B_1, otherwise to v_{f_2}. We define paths $P_1, P_2, \ldots, P_k$ in a plane graph G to be *noncrossing* (for faces f_1 and f_2) if $Image(P_i')$, $1 \le i \le k$, do not cross each other in the plane. Noncrossing paths $P_1, P_2, \ldots, P_k$ are *shortest* if the sum of the lengths of $P_1, P_2, \ldots, P_k$ is minimum. In graph G shown in Figure 3(a), paths P_1 and P_2 cross each other (for the faces f_1 and f_2). On the other hand, the four paths P_1, P_2, P_3, and P_4 shown in Figure 3(b) do not cross each other. This definition is appropriate for the VLSI single-layer routing problem mentioned in Section 1. If each grid edge is of length 1, then the noncrossing paths drawn in thick lines in Figure 1 are shortest.

This paper presents algorithms which necessarily find the shortest noncrossing paths whenever they exist. It is easy to modify the algorithms so that they check the existence of noncrossing paths.

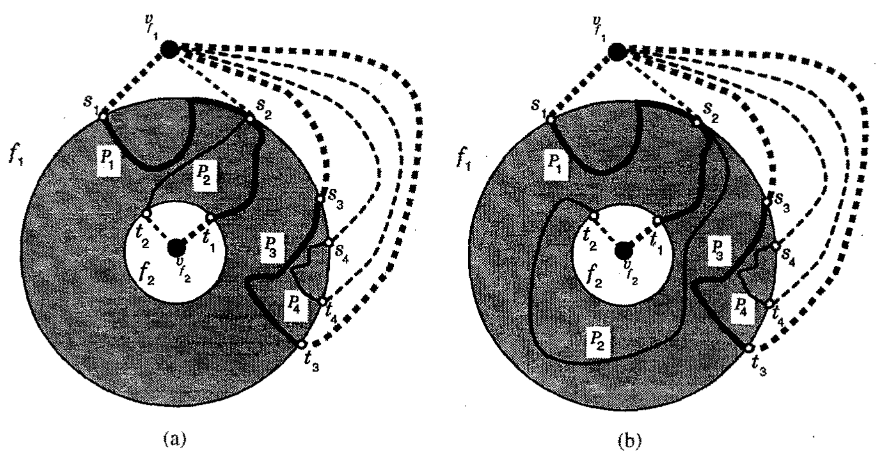


Fig. 3. Crossing paths (a) and noncrossing paths (b).

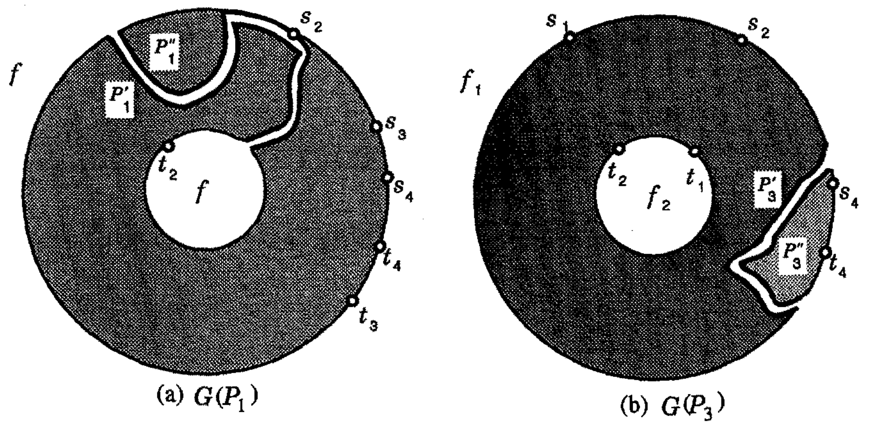


Fig. 4. Slit graphs.

Suppose that path P_1 connecting s_1 to t_1 has been determined. Then paths $P_1, P_2, \ldots, P_k$ are noncrossing (for faces f_1 and f_2) if and only if paths $P_2, P_3, \ldots, P_k$ are noncrossing in a slit graph of G for P_1 defined as follows. A *slit graph* $G(P_1)$ *of* G *for path* P_1 is generated from G by slitting apart path P_1 into two paths P_1' and P_1'', duplicating the vertices and edges of P_1 as follows. Each vertex v in P_1 is replaced by new vertices v' and v''. Each edge (v_j, v_{j+1}) in P_1 is replaced by two parallel edges (v_j', v_{j+1}') and (v_j'', v_{j+1}''). Any edge (v, w) that is not in P_1 but is incident with a vertex v in P_1 is replaced by (v', w) if (v, w) is to the right of a path P_1 going from s_1 to t_1 through $Image(P_1)$, and by (v'', w) if (v, w) is to the left of the path. The operation above is called *slitting* G *along* P_1. If a vertex $v \in V(B_i)$, $i = 1$ or 2, in P_1 is designated as a terminal in G, either v' or v'', that is incident with v_{f_i} in G^+, is designated as a terminal in the slit graph $G(P_1)$. Figures 4(a) and (b) depict the slit graphs $G(P_1)$ for P_1 and $G(P_3)$ for P_3 of G in Figure 3, respectively.

If the slit path P_1 connects two terminals, one on B_1 and the other on B_2, the two faces f_1 and f_2 are merged into a single face in $G(P_1)$ as shown in Figure 4(a). On the other hand, if the slit path P_3 connects two terminals, both on either B_1 or B_2, then $G(P_3)$ is divided into two connected components as shown in Figure 4(b). Furthermore, if there exist k noncrossing paths including P_3 in G, then each pair of terminals different from (s_3, t_3) are in the same connected component of $G(P_3)$. Find noncrossing paths in each connected component of $G(P_3)$. Combine the paths found with P_3, then k noncrossing paths in G can be obtained.

3. The Case When All the Terminals Lie on a Single Face Boundary.

In this section we present an algorithm to find the shortest noncrossing paths for the case when all the terminals are located on the boundary B of a single face f. We assume without loss of generality that f is the outer face of G. A straightforward algorithm for this case is as follows:

```
begin
1. for i = 1 to k do
       begin
2.     find a shortest path P_i connecting s_i and t_i in G;
3.     G := G(P_i)   {slit G along P_i}
       end
end
```

Clearly, each path P_i, $1 \le i \le k$, found by the algorithm above, is a shortest path connecting s_i and t_i in the original graph G. Therefore $P_1, P_2, \dots, P_k$ are shortest noncrossing paths in G. The algorithm runs in time $O(kT(n))$, where $T(n)$ is the time required for finding shortest paths from a single vertex to all other vertices in a plane graph of n vertices. We improve the time complexity to $O(T(n)\log k)$ by separating this case into the following two cases:

Case 1. The terminals $s_1, t_1, s_2, t_2, \dots, s_k, t_k$ appear on B clockwise in this order when we interchange starting terminals s_i and ending terminals t_i and/or indices of terminal pairs if necessary.

Case 2. Otherwise.

We first present Algorithm PATH1(G, S) for Case 1 and then Algorithm PATH2(G,S) for Case 2. PATH1 first decomposes graph G into k subgraphs $G_1, G_2, \dots, G_k$ so that each subgraph G_i contains terminals s_i and t_i. It then finds a shortest path P_i between s_i and t_i in each graph G_i, and finally outputs shortest noncrossing paths $P_1, P_2, \dots, P_k$. For a path or tree P we denote by $P[v, w]$ the path connecting vertices v and w in P.

```
procedure PATH1(G, S);
 begin
1. let T be a shortest path tree containing shortest paths from s_1 to all s_i,
   2 ≤ i ≤ k;
2. for i := 1 to k do
     begin
3.   let G_i be the maximal subgraph of G whose image is in the cycle
     consisting of two paths, the path T[s_i, s_{i+1}] from s_i to s_{i+1} on tree T
     and the path on B counterclockwise going from s_{i+1} to s_i; {s_{k+1} = s_1}
4.   find a shortest path P_i between s_i and t_i in G_i
     end;
5. output {P_i | 1 ≤ i ≤ k}    {the shortest noncrossing paths}
 end;
```

In Figure 5 tree T is drawn in dotted lines and paths P_i in thick lines, and subgraphs G_1, G_2, and G_3 are colored in different gray tones. The following lemma guarantees the correctness of procedure PATH1.

LEMMA 1. *Let G_i, $1 \le i \le k$, be the subgraphs of G found in the procedure* PATH1. *Then graph G_i contains at least one of the shortest paths in G between terminals s_i and t_i.*

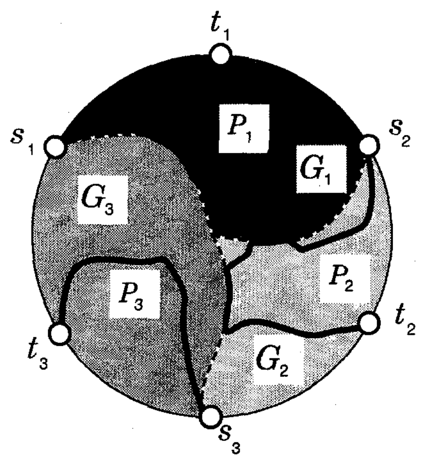


Fig. 5. Illustration for PATH1.

PROOF. Let P_i^* be an arbitrary shortest path connecting s_i and t_i in G. It is sufficient to show that G_i contains a path P_i which is not longer than P_i^*. Let Q_i, $2 \le i \le k$, be the path on T which connects s_1 and s_i on T, and let $Q_1 = Q_2$. If P_i^* does not intersect T, then G_i contains P_i^*. Therefore it may be assumed that P_i^* intersects T. Let a be the vertex on T that appears first on P_i^* going from t_i to s_i. There are two cases to consider.

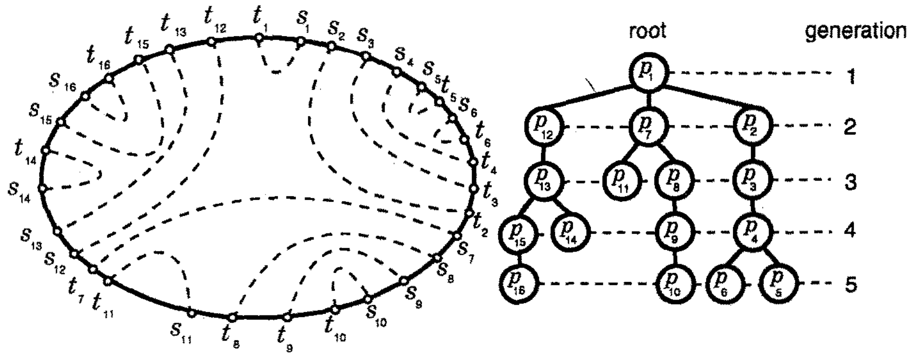
Case 1. a is on Q_i.

In this case the path $T[s_i, a]$ going from s_i to a on T is a shortest path going from s_i to a in G. Therefore $P_i = T[s_i, a] + P_i^*[a, t_i]$ is not longer than P_i^*. Clearly P_i is contained in G_i.

Case 2. Otherwise.

In this case $2 \le i \le k-1$ and vertex a is on Q_{i+1}. Let b be the vertex on Q_{i+1} that appears first on P_i^* going from s_i to t_i, and let c be the vertex on Q_i that appears first on P_i^* going back from b to s_i. (Thus if b is on Q_i, then $b = c$.) Then $G_i + Q_{i+1}$ contains $P_i^*[c, b]$. Therefore, $G_i + Q_{i+1}$ contains the path $P_i = T[s_i, c] + P_i^*[c, b] + T[b, a] + P_i^*[a, t_i]$, and clearly it is not longer than P_i^*. Note that P_i is not necessary a simple path. There exists a simple path P_i' on P_i, which is not longer than P_i and contained in G_i. □

We now consider the execution time of PATH1. All the steps except lines 1 and 4 can be done in time $O(n)$. Line 1, which finds shortest paths from s_1 to all other vertices, can be done in time $O(T(n))$. We claim that line 4 can be executed in time $O(T(n))$ in total. At line 4 each of the k shortest paths is found in a plane subgraph of G bounded by the outer boundary B and tree T. Therefore every edge on T appears in at most two of the subgraphs $G_1, G_2, \ldots, G_k$, and any other edge of G appears in exactly one of them. Thus line 4 can be done in time $O(T(n))$ in total. Therefore the total running time of procedure PATH1 is $O(T(n))$.



(a) (b)

Fig. 6. (a) Terminal pairs. (b) Genealogy tree T_g of height $g = 5$.

We next present Algorithm PATH2 for Case 2 using PATH1. Let $v_1, v_2, \ldots, v_b$ be the vertices on B, and assume that they appear on B clockwise in this order. We may assume without loss of generality that $s_1 = v_1$ and no terminals appear in the subpath of B counterclockwise going from s_1 $(= v_1)$ to t_1. We may assume that, for each terminal pair (s_i, t_i), v_1, s_i, and t_i appear on B clockwise in this order and that $s_1, s_2, \ldots, s_k$ appear on B clockwise in this order. (See Figure 6(a).) For each vertex $v \in V(B)$, *index*(v) denotes the index of v, that is, *index*$(v) = i$ if $v = v_i$. If *index*$(s_i) <$ *index*$(s_j) <$ *index*$(t_j) <$ *index*(t_i), then (s_i, t_i) is an *ancestor* of (s_j, t_j) and (s_j, t_j) is a *descendant* of (s_i, t_i). Note that noncrossing paths do not exist if *index*$(s_i) <$ *index*$(s_j) <$ *index*$(t_i) <$ *index*(t_j). The *parent* (s_l, t_l) of (s_i, t_i) is an ancestor of (s_i, t_i), none of whose descendants is an ancestor of (s_i, t_i). The pair (s_i, t_i) is a *child* of (s_l, t_l). Let T_g be a (genealogy) tree whose nodes correspond to terminal pairs and whose edges correspond to the relation of parent and child. If the terminal pair corresponding to a node p in T_g has a child, then an edge in T_g joins p to the node corresponding to the child. The terminal pair (s_1, t_1) does not have a parent, and is called the *root* of T_g. The *generation* of terminal pair (s_i, t_i) is the depth of the node p_i in T_g corresponding to (s_i, t_i) plus 1. See Figure 6(b). Let g be the maximum generation of nodes. We define similarly the relation of parent and child among paths connecting terminal pairs.

There are two main ideas behind Algorithm PATH2 for Case 2. The first idea is to find shortest noncrossing paths for the terminal pairs of a single generation by using PATH1. Note that such terminal pairs satisfy the requirement for Case 1. We divide G into several components by slitting G along the paths found. For each terminal pair in a component, at least one of the shortest paths connecting the terminal pair in G is contained in the component. Thus we can find shortest noncrossing paths by applying PATH1 to each generation one by one from the first generation to the last. However such a naive implementation of the algorithm above spends time $O(gT(n))$. The second idea is to use the divide-and-conquer method. Our algorithm first finds noncrossing paths for the middle generation, slits the graph along the paths found, and recursively finds noncrossing paths in each connected component. Figure 7 illustrates the idea; Figure 7(a) depicts noncrossing paths for the third generation, that is, the middle generation, in thick

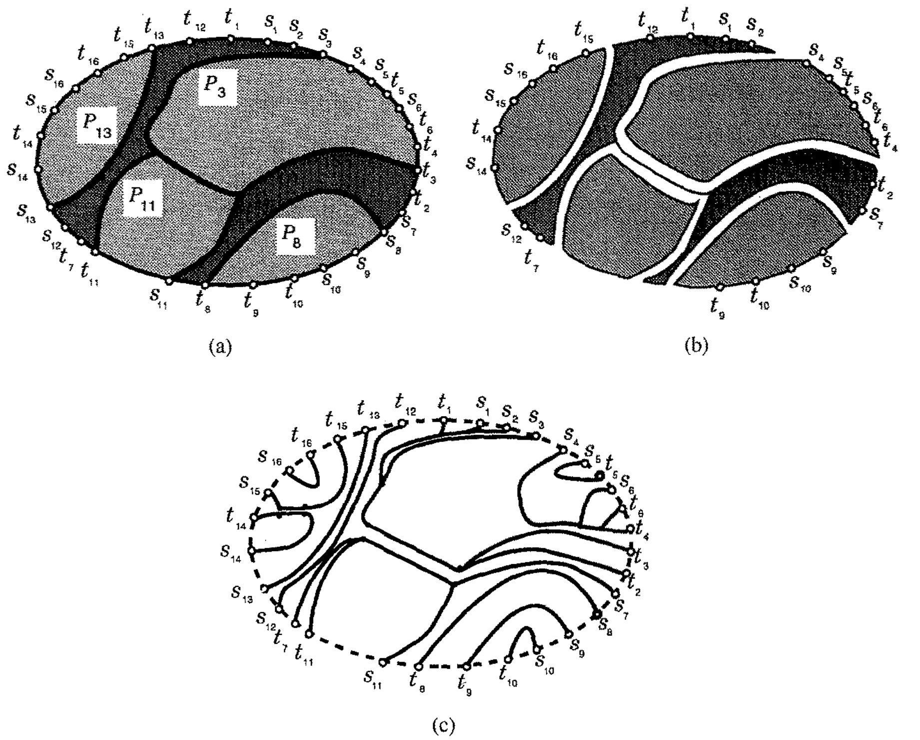


Fig. 7. Illustration for PATH2.

lines; and Figure 7(b) depicts a graph obtained by slitting G along the paths found, where all the terminal pairs of older generations are contained in the dark region and the younger generations in the light region. This way we can obtain a recursive algorithm which runs in time $O(T(n) \log g)$, but we need more definitions to present a formal description of the algorithm.

The *inside* of path P_i connecting terminal pair (s_i, t_i) is the inside of the cycle consisting of P_i and the subpath of B counterclockwise going from t_i to s_i, and is denoted by $in(P_i)$. The *outside* of P_i is the inside of the cycle consisting of P_i and the subpath of B clockwise going from t_i to s_i, and is denoted by $out(P_i)$. The *inside* of a set $\mathcal{P}$ of paths connecting terminal pairs is the union of the insides of paths in $\mathcal{P}$, and is denoted by $in(\mathcal{P})$. The *outside* of $\mathcal{P}$ is the intersection of the outsides of paths in $\mathcal{P}$, and is denoted by $out(\mathcal{P})$.

The output of our algorithms is not a set of k paths but is a set $\mathcal{F}$ of trees which contain the k terminal pairs. The set of paths connecting s_i and t_i, $1 \le i \le k$, on trees in $\mathcal{F}$ are shortest noncrossing paths in G. Since the total number of edges of trees in $\mathcal{F}$ is $O(n)$, the total length of the k paths can be computed by solving the nearest common ancestor problem [GT] for trees in $\mathcal{F}$ total in time $O(n)$ [SAN2]. In Figure 7(c), $\mathcal{F}$ contains 12 trees and each of k $(= 16)$ terminal pairs is contained in one of the trees.

We are now ready to present PATH2.

```
procedure PATH2(G, S);
 begin
 1. let g be the maximum generation of terminal pairs;
 2. F := ∅;
 3. REDUCE(G, [1, g], F)
 end;
```

```
procedure REDUCE(G, [l, h], F);
  begin
 1. if l = h then    {there is only one generation}
      begin
 2.     let S^l be the set of terminal pairs of generation l;
 3.     execute PATH1(G, S^l) and let P_l be the set of found paths;
 4.     F := F ∪ P_l    {detail are mentioned later}
      end
 5. else   {l < h}
      begin
 6.     m := ⌊(l + h)/2⌋;
 7.     let S^m be the set of terminal pairs of generation m;
 8.     execute PATH1(G, S^m), and let P_m be the set of found paths;
 9.     F := F ∪ P_m;    {detail are mentioned later}
10.     let G_in and G_out be the maximal subgraphs of G which are in in(P_m)
        and in out(P_m), respectively;
11.     REDUCE(G_in, [m + 1, h], F);
12.     REDUCE(G_out, [l, m − 1], F)
      end
  end;
```

The running time of PATH2 is dominated by that of REDUCE. REDUCE uses a divide-and-conquer method on generations of terminal pairs. REDUCE first finds shortest noncrossing paths connecting the terminal pairs of the middle generation by using PATH1 in time $O(T(n))$. By slitting G along the determined paths, REDUCE divides the problem into two subproblems, one for the older generations and one for the younger generations. Then these two problems are solved by recursively applying REDUCE. Since the depth of recursive calls is at most $\log g$, we show that REDUCE executed for all subgraphs in the recursive calls of the same depth can be done total in time $O(T(n))$. It suffices to show that every edge in G appears in a constant number, for example at most three, of the subgraphs. We give the detail of the method to divide G and update $\mathcal{F}$ below. (In Figure 7 the edges shared by P_3 and P_{11} appear in three subgraphs.)

REDUCE first finds noncrossing paths $P_1, P_2, \ldots, P_m$ connecting the terminal pairs of the middle generation by using PATH1. Then REDUCE slits G along the paths found. (Figure 8(a) illustrates an example for which $S^m = \{(s_{m_1}, t_{m_1}), (s_{m_2}, t_{m_2})\}$. REDUCE finds Q_{m_1} and Q_{m_2}, and slits G along Q_{m_1} and Q_{m_2} to divide G into three subgraphs G_1, G_{m_1}, and G_{m_2} as shown in Figure 8(b).) Some of the paths Q_j, $1 \le j \le q$, which have been found so far, may appear on the current outer boundary of G. (Figure 8(c)

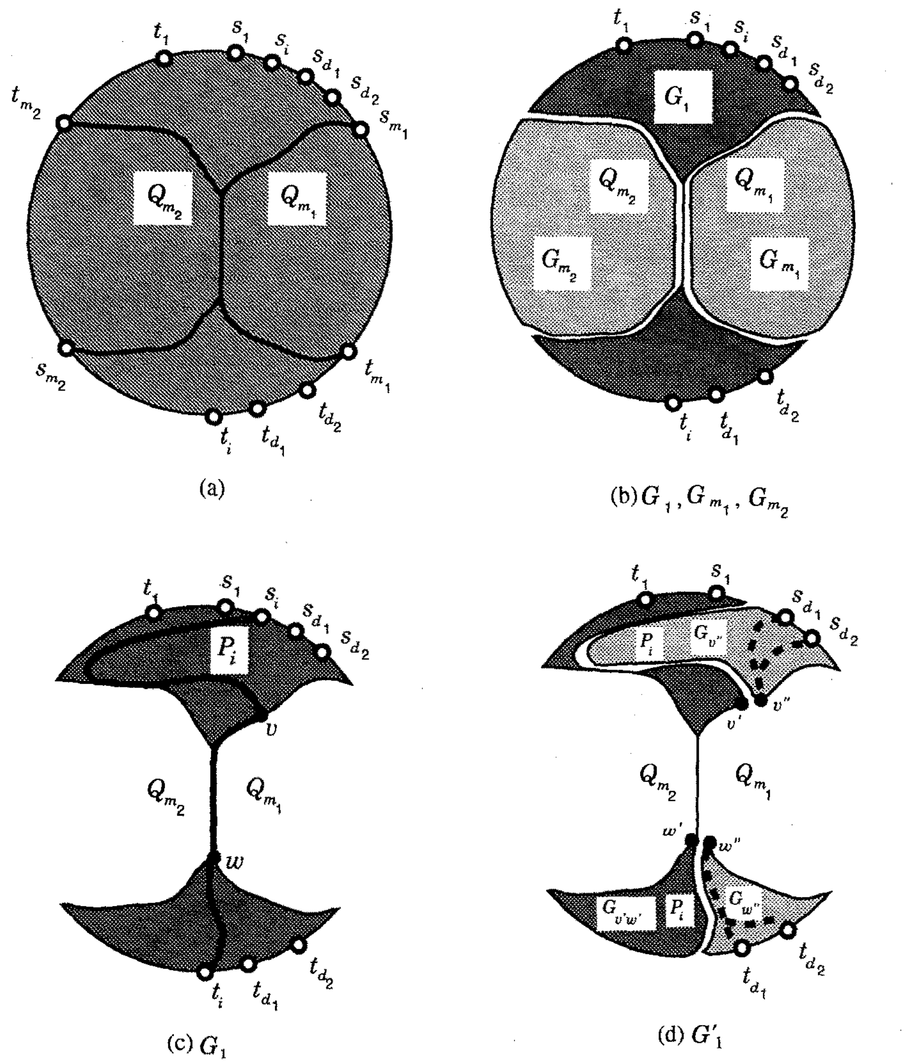


Fig. 8. Illustration for slitting a graph and construction of trees in $\mathcal{F}$.

illustrates the case where Q_{m_1} and Q_{m_2} appear on the outer boundary of G_1.) Note that the edges and vertices of each Q_j have been duplicated. Suppose that G was divided by slitting along the whole path P_i, $1 \le i \le m$. Then the edges shared by P_i and Q_j would be duplicated once more and hence would appear in four or more subgraphs of G. Furthermore, since the vertices of the slit path Q_j have already been included in trees in $\mathcal{F}$, the total number of vertices of trees in $\mathcal{F}$ could not be bounded by $O(n)$. Therefore we divide G and update $\mathcal{F}$ as follows. Assume for simplicity that P_i intersects exactly one of its descendants or ancestors. If P_i is neither an ancestor nor a descendant of Q_j, then we slit G along the whole path P_i. On the other hand, if P_i is an ancestor or a descendant of Q_j, then we slit G along two subpaths of P_i as follows.

Let v be the intersecting vertex of P_i and Q_j that appears first on P_i going from s_i to t_i, and let w be the intersecting vertex of P_i and Q_j that appears last on P_i going from s_i to t_i. It may be assumed that P_i passes through $Q_j[v, w]$. Construct a slit graph $G' = G(P_i[s_i, v] + P_i[w, t_i])$ by slitting G along $P_i[s_i, v]$ and $P_i[w, t_i]$. Update the tree $T \in \mathcal{F}$ containing Q_j by concatenating $P_i[s_i, v]$ and $P_i[w, t_i]$ to $T \in \mathcal{F}$. (Figure 8(d) illustrates a graph G'_1 which is obtained by slitting G_1 along $P_i[s_i, v]$ and $P_i[w, t_i]$. G'_1 consists of three connected components $G_{v''}$, $G_{w''}$, and $G_{v''w''}$. It can be observed from the location of terminal pairs that P_i is an ancestor of Q_{m_1}. Therefore the tree $T \in \mathcal{F}$ which contains Q_{m_1} is updated to a new tree T by concatenating $P_i[s_i, v]$ and $P_i[w, t_i]$ to it.)

When G is slit along $P_i[s_i, v]$ and $P_i[w, t_i]$, v and w are replaced by two new vertices v', v'' and w', w'', respectively. Vertices v' and v'' are contained in distinct connected components of G', and w' and w'' are also contained in distinct connected components of G'. Let p_i be the node of genealogy tree T_g corresponding to P_i, and let p_j be the node of T_g corresponding to Q_j. Let N_{ij} be the set of nodes on the path $T_g[p_i, p_j]$. If $p_d \in N_{ij}$ corresponds to $(s_d, t_d) \in S$, then s_d and t_d are separated into distinct components of G'. (The terminal pair corresponding to $p_d \notin N_{ij}$ are contained in the same connected component of G'.) Since $p_d \in N_{ij}$ is an ancestor or a descendant of p_i, it may be assumed that a shortest path P_d connecting s_d and t_d passes through $P_i[v, w] = T[v, w]$. In each connected component of G' containing such separated terminal pairs, find a shortest path tree T' which is rooted to either v', v'', w', or w'' and contains shortest paths from the root to all separated terminals s_d, t_d. Update $T \in \mathcal{F}$ by concatenating T' to T. Note that the updated tree $T \in \mathcal{F}$ contains paths connecting s_d and t_d. (Figure 8(d) illustrates two shortest path trees found in $G_{v''}$ and $G_{w''}$: one contains shortest paths from v'' to s_{d_1} and s_{d_2}, and the other contains shortest paths from w'' and t_{d_1} and t_{d_2}. We update the tree $T \in \mathcal{F}$ which contains Q_{m_1} and P_i by concatenating the two shortest path trees to T. The two shortest path trees are drawn in dotted lines.) We then divide G' into several subgraphs by slitting G' along the shortest path trees found, and find shortest noncrossing paths in each subgraph by recursively calling REDUCE.

As explained above, if P_i is an ancestor or a descendant of path Q_j which has already been found, then the edges shared by P_i and Q_j are not duplicated. On the other hand, if P_i is neither an ancestor nor a descendant of Q_j, then the edges shared by P_i and Q_j are duplicated again. In this case a path which passes through the shared edges may be found later, but such a path must be an ancestor or a descendant of path P_i or Q_j. Thus every edge in G appears in at most two of the slit paths, and appears in at most three of the subgraphs of the same depth of recursive calls. Moreover, it can be observed that every edge of G appears in at most two trees in $\mathcal{F}$. Thus we can conclude that REDUCE executed for all subgraphs in the recursive calls of the same depth can be done in total in time $O(T(n))$.

Since the depth of recursive calls of REDUCE is $O(\log g)$, the total execution time of REDUCE is $O(T(n) \log g)$. Since $g = O(k)$, PATH2 runs in time $O(T(n) \log k)$ in total. Note that each path P_i found by PATH2 is a shortest path connecting s_i and t_i in G.

4. The Case When All the Terminals Lie on Two Face Boundaries. In this section we present an algorithm for the case when all the terminals lie on two face boundaries B_1 and B_2. For each pair (s_i, t_i) of one terminal on B_1 and the other on B_2, it may be

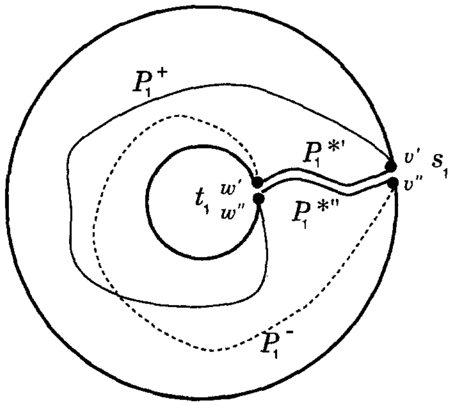


Fig. 9. Slit graph G'_o.

assumed without loss of generality that $s_i \in V(B_1)$ and $t_i \in V(B_2)$. Let

$$\begin{aligned} S_{12} &= \{(s_i, t_i) | s_i \in V(B_1) \text{ and } t_i \in V(B_2)\}, \\ S_1 &= \{(s_i, t_i) | s_i, t_i \in V(B_1)\}, \end{aligned}$$

and

$$S_2 = \{(s_i, t_i) | s_i, t_i \in V(B_2)\}.$$

It may be assumed that $S_{12} \neq \emptyset$: otherwise, shortest noncrossing paths can be easily found by executing PATH2 twice, once for G to find paths for S_1, and then once for the graph obtained by slitting G along the paths found to find paths for S_2.

Let $(s_1, t_1) \in S_{12}$, and let P_1^* be a shortest path between s_1 and t_1 in G. Let G'_0 be the slit graph of G for P_1^*. Then G'_0 has two vertices v' and v'' corresponding to s_1 and two vertices w' and w'' corresponding to t_1. Vertices v', w', w'', and v'' lie on the same face boundary in G'_0 and appear on the boundary clockwise in this order. Let P_1^+ and P_1^- be the two paths in G corresponding to the shortest paths in G'_0 between v' and w'' and between v'' and w', respectively. In Figure 9 P_1^+ and P_1^- are drawn in solid and dotted lines, respectively. Then the following theorem holds, a proof of which is given later in this section.

THEOREM 1. *Let P_1^* be a shortest path connecting $(s_1, t_1) \in S_{12}$ in G. Then G contains shortest noncrossing paths including either P_1^*, P_1^+, or P_1^-.*

Theorem 1 immediately leads to the following algorithm for finding shortest noncrossing paths in G.

procedure PATH(G);
begin
1. find a shortest path P_1^* between s_1 and t_1 in G; $\{(s_1, t_1) \in S_{12}\}$
2. construct the slit graph $G'_0 = G(P_1^*)$, and find paths P_1^+ and P_1^-;

3. $\mathcal{P}_0 := \{P_1^*\}, \mathcal{P}_1 := \{P_1^+\}, \mathcal{P}_2 := \{P_1^-\}$;
   {each $\mathcal{P}_i$, $0 \le i \le 2$, becomes a set of noncrossing paths}
4. construct the slit graph $G_1' = G(P_1^+)$ and the slit graph $G_2' = G(P_1^-)$;
   {all the terminals lie on a single face boundary in G_i', $0 \le i \le 2$.}
5. **for** $i := 0$ **to** 2 **do**
   begin
6. PATH2(G_i', $S - (s_1, t_1)$); {find shortest noncrossing paths in G_i'}
7. add the $k-1$ paths $P_2, P_3, \ldots, P_k$ found to $\mathcal{P}_i$
   end;
8. output, as a solution, one of the sets $\mathcal{P}_0$, $\mathcal{P}_1$, and $\mathcal{P}_2$ whose total length is minimum

end;

The dominating part of the execution time of Algorithm PATH is one for executing PATH2 three times. Therefore the running time of PATH is $O(T(n)\log k)$. Thus we have the following theorem.

THEOREM 2. *Given a plane graph G with k terminal pairs on its two face boundaries, shortest noncrossing paths can be found in time $O(T(n)\log k)$, where $T(n)$ is the time required for finding shortest paths from a single vertex to all other vertices in a plane graph of n vertices.*

A usual shortest path algorithm, that is, Dijkstra's method with a heap, takes time $T(n) = O(n\log n)$ for a plane graph [AHU], [T]. On the other hand Frederickson's method takes time $T(n) = O(n)$ with preprocessing time $O(n\log n)$ [F]. Therefore our algorithm can be implemented to take time $O(n(\log n + \log k)) = O(n\log n)$.

In the remainder of this section we prove Theorem 1. Let $|S_{12}| = l$. Furthermore, let $(s_i, t_i) \in S_{12}$ if $1 \le i \le l$, and $(s_i, t_i) \in S_1 \cup S_2$ otherwise. It may be assumed without loss of generality that terminals $s_1, s_2, \ldots, s_l$ appear on B_1 counterclockwise in this order and $t_1, t_2, \ldots, t_l$ appear on B_2 counterclockwise in this order. For the sake of simplicity, we assume that graph G is embedded in the plane region Σ surrounded by two circles Z_1 with radius 1 and Z_2 with radius $\frac{1}{2}$, both having the center at the origin O of the x–y plane. We may assume without loss of generality that, for each terminal pair (s_i, t_i),

$$Image(s_i) = \left(\cos\left(\frac{2\pi}{l}i\right), \sin\left(\frac{2\pi}{l}i\right)\right)$$

and

$$Image(t_i) = \left(\tfrac{1}{2}\cos\left(\frac{2\pi}{l}i\right), \tfrac{1}{2}\sin\left(\frac{2\pi}{l}i\right)\right),$$

and that $Image(G) \cap (Z_1 \cup Z_2) = \{Image(s_i), Image(t_i) \mid 1 \le i \le l\}$.

Let P be a path going from point a to point b in Σ. Let θ be the total angle (measured counterclockwise) turned through by the line OX when point X moves on P from a to b. Possibly $|\theta| > 2\pi$. We define the (normalized) *angle* $\theta(P)$ of path P by $\theta(P) = \theta/2\pi$. Thus angle $\theta(P_i)$ of path P_i connecting s_i and t_i means the number of rotations of P_i around Z_2. If $P_1, P_2, \dots, P_k$ are noncrossing paths in G, then clearly $\theta(P_1), \theta(P_2), \dots, \theta(P_l)$ are all equal to the same integer.

Note that shortest noncrossing paths are not always *simple* for the case of this section even if every edge has a positive length: each of them may traverse the same vertex or edge more than once, but is necessarily noncrossing itself. Thus in this section a "path" does not always mean a simple one, but is noncrossing itself. The following lemma holds, where *length*(P) denotes the length of path P.

LEMMA 2. *Let P_1^* be a shortest path connecting s_1 and t_1 in G, and let P_i, $1 \le i \le l$, be an arbitrary path connecting s_i and t_i in G. Then there exists in G a path P_i' connecting s_i and t_i such that length$(P_i') \le$ length(P_i) and*

$$\theta(P_i') = \begin{cases} \theta(P_1^*) & \text{if } \theta(P_i) = \theta(P_1^*), \\ \theta(P_1^*) + 1 & \text{if } \theta(P_i) \ge \theta(P_1^*) + 1, \\ \theta(P_1^*) - 1 & \text{if } \theta(P_i) \le \theta(P_1^*) - 1. \end{cases}$$

PROOF. Assume for simplicity that P_i is a simple path. Let $V(P_1^*) \cap V(P_i) = \{v_1, v_2, \dots, v_q\}$, and let $v_1, v_2, \dots, v_q$ appear in this order on $P_1^*[s_1, t_1]$. Denote by $U(P_i)$ the set of vertices v_x, $1 \le x \le q-1$, such that $E(P_1^*[v_x, v_{x+1}]) \cap E(P_i) = \emptyset$. (An example is depicted in Figure 10, where P_1^* is drawn in a thin straight line, P_i in a thick line, $\theta(P_1^*) = 0$, $\theta(P_i) = 2$, $q = 5$, and $U(P_i) = \{v_1, v_2\}$.)

Suppose for a contradiction that the lemma does not hold for a path P_i and furthermore $|U(P_i)|$ is minimum among such paths. If $\theta(P_i) - \theta(P_1^*) = 0$ or ± 1, then clearly path $P_i' = P_i$ satisfies the requirement. Therefore it may be assumed that $\theta(P_i) \ge \theta(P_1^*) + 2$; the proof for the case $\theta(P_i) \le \theta(P_1^*) - 2$ is similar. Then $|U(P_i)| \ge 1$. Let v_a be an arbitrary vertex in $U(P_i)$, and let vertices s_i, v_g, v_h, and t_i appear in this order on P_i where $\{g, h\} = \{a, a+1\}$. Let $Q_i = P_i[s_i, v_g] + P_1^*[v_g, v_h] + P_i[v_h, t_i]$, then clearly Q_i is a path connecting s_i and t_i and *length*$(Q_i) \le$ *length*(P_i). Since $P_i[v_g, v_h] + P_1^*[v_h, v_g]$ is a cycle, $|\theta(P_i[v_g, v_h]) - \theta(P_1^*[v_g, v_h])| = 0$ or 1, and hence $\theta(Q_i) \ge \theta(P_i) - 1 \ge \theta(P_1^*) + 1$.

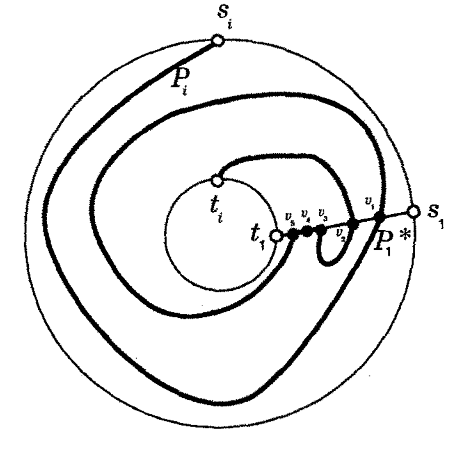


Fig. 10. Illustration for proof of Lemma 2.

Therefore the lemma does not hold for Q_i either. However, $|U(Q_i)| < |U(P_i)|$, a contradiction. □

In Lemmas 3–5 following let $\theta = 0, \pm 1, \pm 2, \ldots$, and let P_1 be the shortest one among the paths in G which connect s_1 and t_1 and have angle θ. We have the following lemma.

LEMMA 3. *Let u and w be two vertices on P_1, and let Q be a path between u and w in G such that $\theta(Q[u, w]) = \theta(P_1[u, w])$. Then length($Q$) $\geq$ length($P_1[u, w]$).*

PROOF. Assume for simplicity that path P_1 is simple. Let the intersecting vertices $v_1 (= u), v_2, \ldots, v_q (= w)$ of P_1 and Q appear on $Q[u, w]$ in this order. For each x, $1 \leq x \leq q$, let $r_x = \theta(Q[v_1, v_x]) - \theta(P_1[v_1, v_x])$. Then the sequence of integers $r_1, r_2, \ldots, r_q$ satisfies

$$\begin{cases} r_1 = r_q = 0, & \\ r_x - r_{x+1} = 0,\ \pm 1 & \text{for every } x, \quad 1 \leq x \leq q-1. \end{cases}$$

Denote by $U(Q)$ the set of vertices v_x, $1 \leq x \leq q-1$, such that $E(Q[v_x, v_{x+1}]) \cap E(P_1) = \emptyset$. For a cycle C, denote by $\mathcal{I}(C)$ the set of paths $Q[v_x, v_{x+1}]$, $1 \leq x \leq q-1$, which are properly inside C except for the ends.

Suppose for a contradiction that the lemma does not hold for a path Q and furthermore $|U(Q)|$ is minimum among such paths. Then $U(Q) \neq \emptyset$; otherwise Q is a path on P_1 and hence the lemma holds.

We claim that $r_x \neq r_{x+1}$ for every $v_x \in U(Q)$. Let $r_x = r_{x+1}$ for a vertex $v_x \in U(Q)$, and let $|\mathcal{I}(C_x)|$ be minimum among such vertices where $C_x = Q[v_x, v_{x+1}] + P_1[v_{x+1}, v_x]$. If there is a path $Q[v_y, v_{y+1}]$ in $\mathcal{I}(C_x)$, then clearly $r_y = r_{y+1}$ and $|\mathcal{I}(C_y)| < |\mathcal{I}(C_x)|$ where $C_y = Q[v_y, v_{y+1}] + P_1[v_{y+1}, v_y]$, a contradiction. Thus $\mathcal{I}(C_x) = \emptyset$. Then path $P_1' = P_1[s_1, v_g] + Q[v_g, v_h] + P_1[v_h, v_q]$ has angle θ and is noncrossing itself, where $\{g, h\} = \{x, x+1\}$ and s_1, v_g, v_h, and t_1 appear on P_1 in this order. Since length(P_1') $\geq$ length(P_1), $Q[v_g, v_h] \geq P_1[v_g, v_h]$. Therefore path $Q' = Q[v_1, v_x] + P_1[v_x, v_{x+1}] + Q[v_{x+1}, v_q]$ satisfies length(Q') $\leq$ length(Q) and $\theta(Q') = \theta(P_1[v_1, v_q])$. Since $|U(Q')| = |U(Q)| - 1$, the lemma holds for Q', that is, length(Q') $\geq$ length($P_1[v_1, v_q]$), and hence length(Q) $\geq$ length($P_1[v_1, v_q]$), a contradiction.

Thus $r_i > 0$ for some integer i, $2 \leq i \leq q-1$. Considering r_i a maximal positive number, we know that there are two integers a and b such that $3 \leq a+2 \leq b \leq q$ and $r_a = r_b = r_i - 1$ for every i, $a < i < b$. Since $r_x \neq r_{x+1}$ for every $v_x \in U(Q)$, $v_i \notin U(Q)$ for every i, $a < i < b-1$. Therefore $Q[v_{a+1}, v_{b-1}]$ is a path on P_1, and $C_{ab} = Q[v_a, v_b] + P_1[v_b, v_a]$ is a cycle. Choose a and b so that $|\mathcal{I}(C_{ab})|$ is minimum. If $\mathcal{I}(C_{ab}) = \emptyset$, then an argument similar to above leads to a contradiction. Therefore it may be assumed that there is a path $Q[v_c, v_{c+1}]$ in $\mathcal{I}(C_{ab})$. Clearly, $r_c \neq r_{c+1}$. Let $r_c < r_{c+1}$; the proof for the case $r_c > r_{c+1}$ is similar. Then there is an integer d such that $c+2 \leq d \leq q$ and $r_c = r_d = r_i - 1$ for every i, $c < i < d$. Since cycle $C_{cd} = Q[v_c, v_d] + P_1[v_d, v_c]$ is inside C_{ab}, $|\mathcal{I}(C_{cd})| < |\mathcal{I}(C_{ab})|$, contrary to the selection of a and b. □

Using Lemma 3, we can have the following two lemmas.

LEMMA 4. *Let $P_i, 2 \le i \le l$, be a path which connects s_i and t_i and has angle θ. Then there exists a path P'_i connecting s_i and t_i such that*:

(a) *P'_i does not cross P_1.*
(b) *$\theta(P'_i) = \theta(P_i) = \theta$ and $length(P'_i) \le length(P_i)$.*

PROOF. It may be assumed that P_i crosses P_1. Let the intersecting vertices $v_1, v_2, \ldots, v_q$ of P_i and P_1 appear on $P_i[s_i, t_i]$ in this order. For each x, $1 \le x \le q$, let $r_x = \theta(P_i[v_1, v_x]) - \theta(P_1[v_1, v_x])$. Then $r_q = 0, \pm 1$.

Consider first the case $r_q = 0$. In this case by Lemma 3 $length(P_1[v_1, v_q]) \le length(P_i[v_1, v_q])$, and hence path $P'_i = P_i[s_i, v_1] + P_1[v_1, v_q] + P_i[v_q, t_i]$ satisfies $length(P'_i) \le length(P_i)$. Clearly, $\theta(P'_i) = \theta$ and P'_i does not cross P_1.

Consider next the case $r_q = 1$; the proof for the case $r_q = -1$ is similar. In this case $r_a = 0$ and $r_{a+1} = 1$ for an integer a, $1 \le a \le q-1$. Since $\theta(P_1[v_1, v_a]) = \theta(P_i[v_1, v_a])$ and $\theta(P_1[v_{a+1}, v_q]) = \theta(P_i[v_{a+1}, v_q])$, by Lemma 3 $length(P_1[v_1, v_a]) \le length(P_i[v_1, v_a])$ and $length(P_1[v_{a+1}, v_q]) \le length(P_i[v_{a+1}, v_q])$. Therefore path $P'_i = P_i[s_i, v_1] + P_1[v_1, v_a] + P_i[v_a, v_{a+1}] + P_1[v_{a+1}, v_q] + P_i[v_q, t_i]$ satisfies $length(P'_i) \le length(P_i)$. Clearly, $\theta(P'_i) = \theta$ and P'_i does not cross P_1. □

Let $(s_i, t_i) \in S_1 \cup S_2$, i.e., $l+1 \le i \le k$, and let P_i be a path between s_i and t_i. If $(s_i, t_i) \in S_1$, then let s_1, s_i, t_i appear on B_1 counterclockwise in this order and let A be the path on B_1 clockwise going from t_i to s_i. If $(s_i, t_i) \in S_2$, then let t_1, s_i, t_i appear on B_2 counterclockwise in this order and let A be the path on B_2 clockwise going from t_i to s_i. Let C_{P_i} be the cycle consisting of two paths, $P_i[s_i, t_i]$ and A. Observe that if P_i does not cross a path connecting s_1 and t_1, then cycle C_{P_i} does not contain face f_2 inside. Conversely the following lemma holds.

LEMMA 5. *If cycle C_{P_i} does not contain f_2 inside, then there is a path P'_i between s_i and t_i which satisfies $length(P'_i) \le length(P_i)$ and does not cross P_1.*

PROOF. It may be assumed that P_i crosses P_1. Let the intersecting vertices $v_1, v_2, \ldots, v_q$ of P_i and P_1 appear on $P_i[s_i, t_i]$ in this order. For each x, $1 \le x \le q$, let $r_x = \theta(P_i[v_1, v_x]) - \theta(P_1[v_1, v_x])$. Let $D = P_i[v_q, t_i] + A[t_i, s_i] + P_i[s_i, v_1]$, then $C_{P_i} = P_i[v_1, v_q] + D$. Since cycle C_{P_i} does not contain f_2 inside, $\theta(P_i[v_1, v_q]) + \theta(D) = 0$. Since $P_1[v_1, v_q] + D$ is a cycle, $|\theta(P_1[v_1, v_q]) + \theta(D)| \le 1$. Therefore we have $|r_q| \le 1$.

Consider first the case $r_q = 0$. Then by Lemma 3 $length(P_1[v_1, v_q]) \le length(P_i[v_1, v_q])$. Therefore path $P'_i = P_i[s_i, v_1] + P_1[v_1, v_q] + P_i[v_q, t_i]$ satisfies $length(P'_i) \le length(P_i)$, and clearly P'_i does not cross P_1.

Consider next the case $r_q = 1$; the proof for the case $r_q = -1$ is similar. In this case there is an integer a such that $1 \le a \le q-1$, $r_a = 0$, and $r_{a+1} = 1$. Since $length(P_1[v_1, v_a]) \le length(P_i[v_1, v_a])$ and $length(P_1[v_{a+1}, v_q]) \le length(P_i[v_{a+1}, v_q])$, $P'_i = P_i[s_i, v_1] + P_1[v_1, v_a] + P_i[v_a, v_{a+1}] + P_1[v_{a+1}, v_q] + P_i[v_q, t_i]$ satisfies $length(P'_i) \le length(P_i)$. Furthermore, P'_i does not cross P_1. □

Furthermore, the following lemma clearly holds.

LEMMA 6. *Let P_1 be an arbitrary path in G connecting s_1 and t_1. Then G contains paths P_i, $2 \le i \le k$, such that:*

(a) *$P_1, P_2, \cdots, P_k$ are noncrossing in G.*
(b) *Each path P_i, $2 \le i \le k$, is a shortest one among the paths in G which connect s_i and t_i and does not cross P_1.*

PROOF. Construct $G(P_1)$ by slitting G along P_1. Then two faces f_1 and f_2 are merged into a same single face f in $G(P_1)$ and all terminals lie on the boundary of f (see Figure 4(a)). Let $P_2, P_3, \ldots, P_k$ be the shortest noncrossing paths in $G(P_1)$ obtained by applying procedure PATH2 for $G(P_1)$. Then $P_1, P_2, \ldots, P_k$ are noncrossing in G. Furthermore, each path P_i, $2 \le i \le k$, is shortest among the paths in $G(P_1)$ which connect s_i and t_i, and hence P_i is shortest among the paths in G which connect s_i and t_i and do not cross P_1. □

We are now ready to prove Theorem 1.

PROOF OF THEOREM 1. Let $P_1, P_2, \ldots, P_k$ be arbitrary shortest noncrossing paths in G. Clearly, $\theta(P_1) = \theta(P_2) = \cdots = \theta(P_l)$. By Lemma 4 applied for P_1^* and each P_i, $1 \le i \le l$, we know that there exists a path P_i' connecting s_i and t_i such that $length(P_i') \le length(P_i)$ and

$$\theta(P_i') = \begin{cases} \theta(P_1^*) & \text{if } \theta(P_i) = \theta(P_1^*), \\ \theta(P_1^*) + 1 & \text{if } \theta(P_i) \ge \theta(P_1^*) + 1, \\ \theta(P_1^*) - 1 & \text{if } \theta(P_i) \le \theta(P_1^*) - 1. \end{cases}$$

Note that $\theta(P_1') = \theta(P_2') = \cdots = \theta(P_l')$ but $P_1', P_2', \ldots, P_l'$ may cross each other.

Clearly, $\theta(P_1^+) = \theta(P_1^*) + 1$ and $\theta(P_1^-) = \theta(P_1^*) - 1$. Similarly as in the proof of Lemma 2, it can be proved that path P_1^+ is shortest among the paths in G which connect s_1 and t_1 and have angle $\theta(P_1^*) + 1$, and path P_1^- is shortest among the paths in G which connect s_1 and t_1 and have angle $\theta(P_1^*) - 1$.

Let P_1'' be

$$P_1'' = \begin{cases} P_1^* & \text{if } \theta(P_1) = \theta(P_1^*), \\ P_1^+ & \text{if } \theta(P_1) \ge \theta(P_1^*) + 1, \\ P_1^- & \text{if } \theta(P_1) \le \theta(P_1^*) - 1, \end{cases}$$

and let $\theta = \theta(P_1'') = \theta(P_1')$. Then by Lemmas 4 and 5 there exist paths $P_2'', P_3'', \ldots, P_k''$ such that each P_i'', $2 \le i \le k$, does not cross P_1'' and satisfies

$$length(P_i'') \le \begin{cases} length(P_i') & \text{if } 2 \le i \le l, \\ length(P_i) & \text{if } l + 1 \le i \le k. \end{cases}$$

By Lemma 6 G contains noncrossing paths $P_1''' = P_1'', P_2''', P_3''', \ldots, P_k'''$ such that $length(P_i''') \le length(P_i'')$, $1 \le i \le k$. $P_1''', P_2''', \ldots, P_k'''$ are shortest noncrossing paths in G since $length(P_i''') \le length(P_i)$ for every i, $1 \le i \le k$. □

5. Optimal Noncrossing Paths. In this section we show that, slightly modifying the algorithms in the preceding sections, "optimal" noncrossing paths can be found.

Let $P_1, P_2, \ldots, P_k$ be noncrossing paths in a plane graph G, where P_i connects terminals s_i and t_i. Denote the length of P_i by l_i. Let $f(l_1, l_2, \ldots, l_k)$ be an arbitrary (objective) function which is nondecreasing with respect to each variable l_i. We call noncrossing paths $P_1, P_2, \ldots, P_k$ minimizing $f(l_1, l_2, \ldots, l_k)$ *optimal noncrossing paths* (with respect to the objective function f).

EXAMPLE 1. The shortest noncrossing paths are optimal ones minimizing the objective function $f = \sum_{i=1}^{k} l_i$. Clearly, f is nondecreasing with respect to each l_i.

EXAMPLE 2. If all the paths (wires) have the same width, then the shortest noncrossing paths correspond to a routing minimizing the area required by wires. On the other hand, if the paths have various widths, say P_i has width w_i, then optimal paths minimizing $f = \sum_i w_i l_i$ correspond to a routing minimizing the area. This function f is also nondecreasing with respect to each l_i.

EXAMPLE 3. Noncrossing paths minimizing $f = \max\{l_1, l_2, \ldots, l_k\}$ are desirable when one wishes to minimize the time delay in wires. Such an f is also nondecreasing with respect to each l_i.

There are two cases:

Case 1. All the terminals lie on a single face boundary.

The algorithm in Section 3 finds noncrossing paths $P_1, P_2, \cdots, P_k$ such that P_i, $1 \leq i \leq k$, is a truly shortest path between s_i and t_i in G. Since f is nondecreasing with respect to each length l_i, these paths $P_1, P_2, \ldots, P_k$ minimize f and hence are optimal noncrossing paths.

Case 2. All the terminals lie on two face boundaries.

Similarly as the proof of Theorem 1, one can prove the following theorem.

THEOREM 3. *Let P_1^* be a shortest path connecting $(s_1, t_1) \in S_{12}$ in G. Then G contains optimal noncrossing paths including either P_1^*, P_1^+, or P_1^-.*

Hence optimal noncrossing paths can be found by procedure PATH if line 8 is replaced by:

8. output, as a solution, one of the sets $\mathcal{P}_0$, $\mathcal{P}_1$, and $\mathcal{P}_2$ that minimizes the objective function f

Thus, if the function f for given noncrossing paths can be evaluated in $O(n \log n)$ time, then optimal paths can be found in $O(n \log n)$ time.

6. Conclusion. In this paper we presented an efficient algorithm for finding shortest or optimal noncrossing paths for the case where terminal pairs are located on two specified face boundaries of a plane graph, and proved that the running time is $O(n \log n)$. Furthermore, it is rather straightforward to modify our sequential algorithm to an NC parallel algorithm which finds shortest or optimal noncrossing paths in polylog time using a polynomial number of processors. Note that there are NC parallel algorithms for the shortest path problem on general or planar graphs [J], [K]. We are now extending the algorithm to a more general case where terminals lie on three or more face boundaries and a case where terminals lie on the plane with several rectangular obstacles.

Acknowledgments. We wish to thank Professor Tetsuo Asano of Osaka Electro-Communication University and Professor D. T. Lee of Northwestern University for suggesting the noncrossing path problem. We also wish to thank the reviewers for their very useful suggestion to improve the presentation of the paper.

References

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

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

[F] G. N. Frederickson, Fast algorithms for shortest paths in planar graphs, with applications, *SIAM J. Comput.*, **16** (1987), 1004–1022.

[GT] H. N. Gabow and R. E. Tarjan, A linear-time algorithm for a special case of disjoint set union, *J. Comput. System Sci.*, **30** (1985), 209–221.

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

[K] P. N. Klein, A linear processor polylog-time algorithm for shortest path in planar graphs, *Proc. 34th Symp. on Foundations of Computer Science*, pp. 259–270, 1993.

[KL] M. R. Kramer and J. van Leewen, Wire-routing is NP-complete, Report No. RUU-CS-82-4, Department of Computer Science, University of Utrecht, Utrecht, 1982.

[LSYW] D. T. Lee, C. F. Shen, C. D. Yang, and C. K. Wong, Non-crossing path problems, Manuscript, Dept. of EECS, Northwestern University, 1991.

[L] J. F. Lynch, The equivalence of theorem proving and the interconnection problem, *ACM SIGDA Newsletter*, **5**(3) (1975), 31–36.

[SAN1] H. Suzuki, T. Akama, and T. Nishizeki, Algorithms for finding internally-disjoint paths in a planar graph, *Trans. IECE*, **J71-A**(10) (1988), 1906–1916 (in Japanese).

[SAN2] H. Suzuki, T. Akama, and T. Nishizeki, Finding Steiner forests in planar graphs, *Proc. First SIAM–ACM Symp. on Discrete Algorithms*, pp. 444–453, 1990.

[T] R. E. Tarjan, *Data Structures and Network Algorithms*, SIAM, Philadelphia, PA, 1983.

[TSN] J. Takahashi, H. Suzuki, and T. Nishizeki, Algorithms for finding noncrossing paths with minimum total length in plane graphs, *Proc. ISAAC '92*, Lecture Notes in Computer Science, vol. 650, Springer-Verlag, Berlin, pp. 400–409, 1992.