Algorithms for Finding Non-Crossing Paths with Minimum Total Length in Plane Graphs

Jun-ya Takahashi, Hitoshi Suzuki and Takao Nishizeki

Department of Information Engineering
Faculty of Engineering, Tohoku University
Sendai 980, Japan

Abstract. Let G be an undirected plane graph with non-negative edge length, and let k terminal pairs lie on two specified face boundaries. This paper presents an algorithm for finding k "non-crossing paths" in G, each connecting a terminal pair, whose total length is minimum. Here "non-crossing 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.

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 has many practical applications such as VLSI layout design. An edge in the graph corresponds to a routing region in VLSI chip. The problem is NP-complete [Lyn,KL], and so it is very unlikely that there exists a polynomial-time algorithm to solve the problem. If, however, two or more wires may pass through a single routing region, then the problem can be reduced to the shortest "non-crossing" path problem, where "non-crossing" paths may share common vertices or edges but do not cross each other in the plane. The shortest non-crossing path problem is expected to be solvable in polynomial time at least for a restricted case, for example, a case when all terminals are located on boundaries of a constant number of faces in a plane graph.

In this paper we give an $O(n \log n)$ algorithm which finds shortest non-crossing paths in a plane graph for the case when all the terminals are located on two specified face boundaries, where n is the number of vertices in G. For the same case Suzuki, Akama and Nishizeki [SAN] 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 at all. 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).

In Section 2 we give a formal description of the problem and define several terms. In Section 3 we present an algorithm for the case where all terminals lie on a single face boundary. In Section 4 we give an algorithm for the case where all terminals lie on two face boundaries. Section 5 is a conclusion.

2. Preliminaries

In this section we give a formal description of the non-crossing path problem and define several terms. We denote by $G = (V, E)$ the graph consisting of a vertex set V and an edge set E. We sometimes denote by $V(G)$ and $E(G)$ the

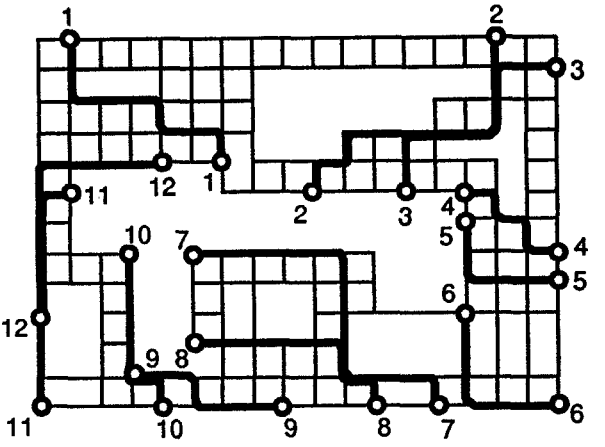

Fig.1. Shortest non-crossing paths in a grid graph.

vertex and edge sets of G, respectively. We assume that G is a 2-connected undirected plane graph and every edge in G has a non-negative 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. 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 k be the number of terminal pairs. In this paper we do not assume that k is a constant. Suppose that all the terminals are located on boundaries B_1 and B_2 of two specified faces f_1 and f_2. Assume for simplicity that $V(B_1) \cap V(B_2) = \phi$ and all terminals are distinct from each other.

Let $P_1, P_2, \cdots, P_k$ 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; the resulting graph is G^+. Let $P_i', 1 \leq i \leq k$, be a path (or a cycle) in G^+ obtained from P_i by adding two new edges: one joining s_i to v_{f_1} or v_{f_2}, and the other joining t_i to v_{f_1} or v_{f_2}. We define paths $P_1, P_2, \cdots, P_k$ in a plane graph G to be *non-crossing* (for faces f_1 and f_2) if $Image(P_i'), 1 \leq i \leq k$, do not cross each other in the plane. Figure 2(a) depicts non-crossing paths P_1, P_2, P_3 and P_4. Non-crossing paths $P_1, P_2, \cdots, P_k$ are *shortest* if the sum of the lengths of $P_1, P_2, \cdots, P_k$ is minimum.

This paper presents an algorithm to solve the following *non-crossing path problem.*

Non-crossing path problem: Find shortest non-crossing paths, each connecting a terminal pair on two specified face boundaries in a plane graph G.

Figure 1 depicts shortest non-crossing paths in a grid graph where each edge has length 1.

Suppose that path P_1 connecting s_1 and t_1 has been decided. Then paths $P_1, P_2, \cdots, P_k$ are non-crossing (for faces f_1 and f_2) if paths $P_2, P_3, \cdots, P_k$ are non-crossing 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 (See Figure 2). Each vertex

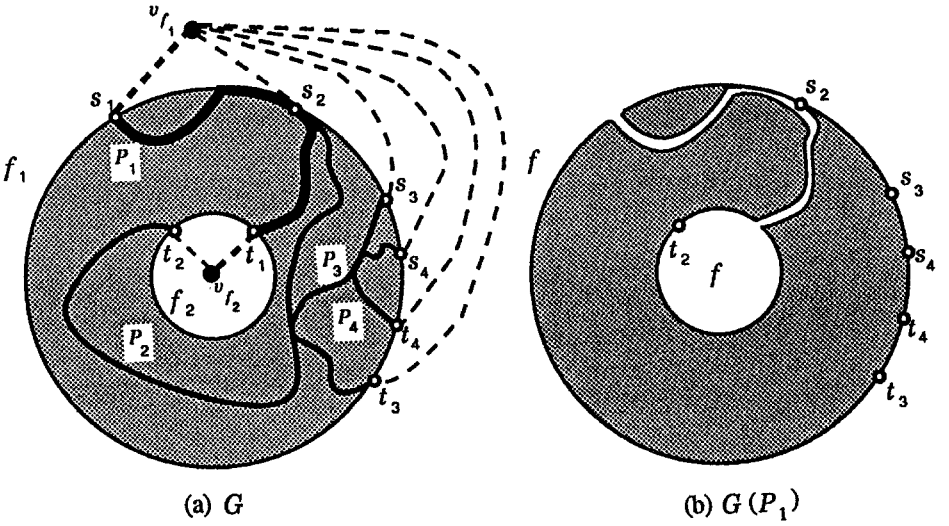

Fig.2. Graph G and a slit graph.

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 $G(P_1)$.

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

In this section we present an algorithm to solve the non-crossing path problem for the case when all the terminals are located on the boundary B of a single face f. We assume w.l.o.g. that f is the outer face of G. Let S be the set of terminal pairs. We separate this case into the following two cases:

CASE 1: the terminals $s_1, t_1, s_2, t_2, \cdots, s_k, t_k$ 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 an algorithm PATH1(G, S) for Case 1 and then an algorithm PATH2(G,S) for Case 2. PATH1 first decomposes the graph G into k subgraphs $G_1, G_2, \cdots, G_k$ 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 the k shortest non-crossing paths $P_1, P_2, \cdots, P_k$. Denote by $P[v, w]$ the path connecting vertices v and w in a path or tree P.

procedure PATH1(G, S);
begin
1. let T be a shortest path tree containing shortest paths from s_1 to all s_i, $2 \leq i \leq k$;
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 $\{P_i | 1 \leq i \leq k\}$ {the shortest non-crossing paths}
end;

In Figure 3 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.

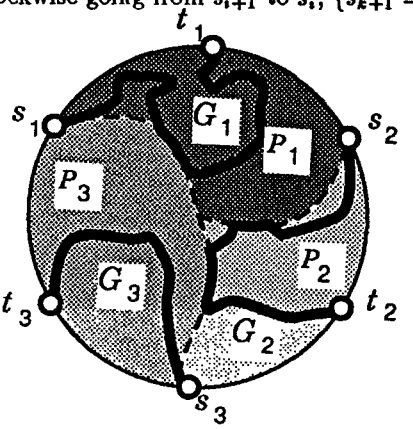

Fig.3. Illustration for PATH1.

LEMMA 1. Let G_i, $1 \leq i \leq k$, be the subgraphs 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.

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 $O(T(n))$, 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 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 region of G bounded by B and tree T. Therefore every edge on T appears in at most two of the subgraphs $G_1, G_2, \cdots, G_k$, 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 $O(T(n))$.

We next present an algorithm PATH2 for Case 2 using PATH1. Let $v_1, v_2, \cdots, v_b$ be the vertices on B, and assume that they appear on B clockwise in this order. We may assume w.l.o.g. 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 $s_1, s_2, \cdots, s_k$ appear on B clockwise in this order (see Figure 4(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 non-crossing paths do not exist if $index(s_i) < index(s_j) < index(t_i) < index(t_j)$. Let (s_l, t_l) be the ancestor of (s_i, t_i) having the maximum index. Then (s_l, t_l) is the *parent* of (s_i, t_i), and (s_i, t_i) is a *child* of (s_l, t_l). Let T_g be the (genealogy) tree whose nodes correspond to terminal pairs and edges correspond to the relation of parent and child. Thus, 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 4(b).

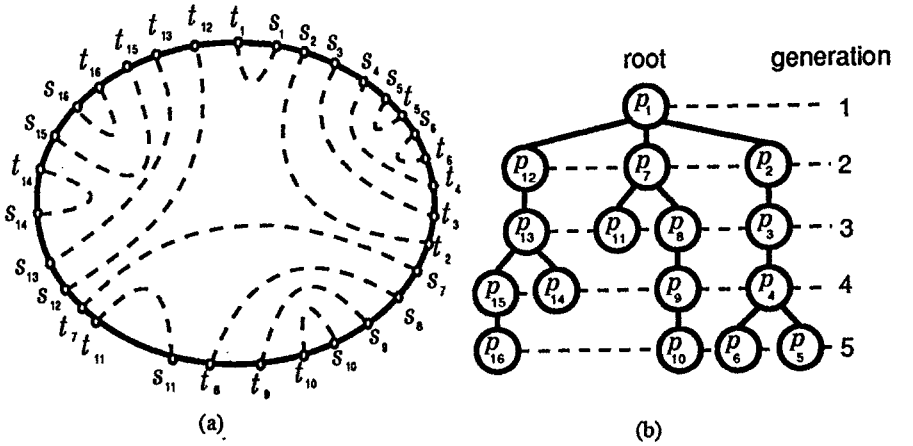

Fig.4. (a) Terminal pairs, and (b) genealogy tree T_g.

There are two ideas in the algorithm PATH2 for Case 2. The first idea is to find non-crossing paths for the terminal pairs of the same 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 found paths. 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 non-crossing 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 $O(kT(n))$. The second idea is to use the divide-and-conquer method. Our algorithm first finds non-crossing paths for the middle generation, slits the graph along the found paths, and recursively finds non-crossing paths in each connected component.

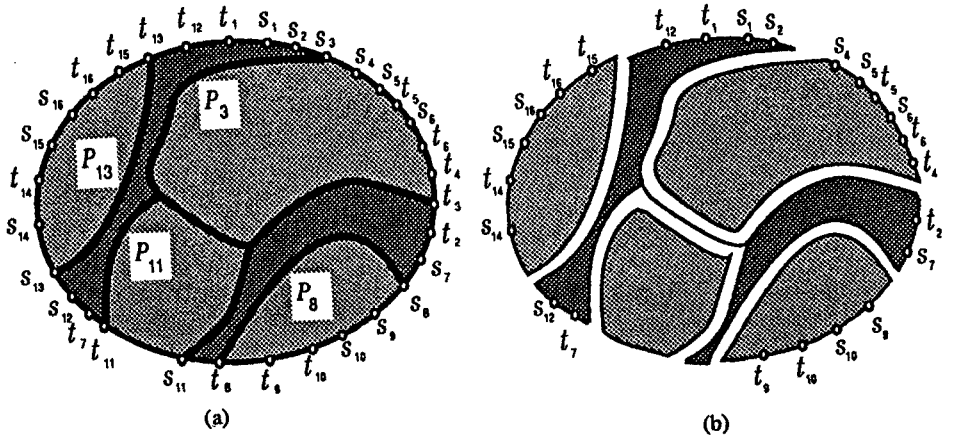

Fig.5. Illustration for PATH2.

Figure 5 illustrates the idea; Figure 5(a) depicts non-crossing paths for the third generation, that is, the middle generation in thick lines; and Figure 5(b) depicts a graph obtained by slitting G along the found paths, where all the terminal pairs of older generations are contained in the dark region and the younger in the light region. In such a way we can obtain an algorithm whose running time is $O(T(n)\log k)$, but 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 $t_i, 1 \leq i \leq k$, on trees in $\mathcal{F}$ is a solution of the non-crossing path problem. Since the total number of vertices of trees in $\mathcal{F}$ is $O(n)$, we can compute the total length of the k paths by solving the nearest common ancestor problem [GT] for trees in $\mathcal{F}$ total in time $O(n)$ [SAN].

```
procedure PATH2(G, S);
    begin
1.      let g be the maximum generation of terminal pairs;
2.      F := φ;
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
            end
5.      else
            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;
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 non-crossing paths connecting the terminal pairs of the middle generation by using PATH1 in time $O(T(n))$. Then REDUCE slits G along the found paths, divides the problem into two, one for older generations and the other for younger generations, and recursively solves the problem by calling REDUCE itself. Clearly

the depth of the recursive calls is at most $\log k$. Procedure PATH1 executed for all subgraphs in the recursive calls of the same depth can be done in time $O(T(n))$ in total, because one can implement REDUCE so that every edge in G appears at most three of the subgraphs. Therefore the total execution time of REDUCE is $O(T(n)\log k)$. Thus we can conclude that the shortest non-crossing paths $P_1, P_2, \cdots, P_k$ can be found in $O(T(n)\log k)$ time for Case 2. Note that each path P_i is a shortest path connecting s_i and t_i in G.

4. The Case where 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 terminal pair (s_i, t_i) of one on B_1 and the other on B_2, one may assume w.l.o.g. that $s_i \in V(B_1)$ and $t_i \in V(B_2)$. Let

$$S_{12} = \{(s_i, t_i) | s_i \in V(B_1) \text{ and } t_i \in V(B_2)\},$$
$$S_1 = \{(s_i, t_i) | s_i, t_i \in V(B_1)\} \text{ and}$$
$$S_2 = \{(s_i, t_i) | s_i, t_i \in V(B_2)\}.$$

One may assume that $S_{12} \neq \emptyset$: otherwise, the non-crossing path problem can be easily solved 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 found paths to find paths for S_2.

Assume that $(s_i, t_i) \in S_{12}$ if $1 \leq i \leq l$, and $(s_i, t_i) \in S_1 \cup S_2$ otherwise. One may assume w.l.o.g. that terminals $s_1, s_2, \cdots, s_l$ appears on B_1 counterclockwise in this order and $t_1, t_2, \cdots, t_l$ appears 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 C_1 with radius 1 and C_2 with radius 1/2 both having the center at the origin O of the xy-plane. We may assume w.l.o.g. that, for each terminal pair (s_i, t_i), $Image(s_i) = (\cos(\frac{2\pi}{l}i), \sin(\frac{2\pi}{l}i))$ and $Image(t_i) = (\frac{1}{2}\cos(\frac{2\pi}{l}i), \frac{1}{2}\sin(\frac{2\pi}{l}i))$, and that $Image(G) \cap (C_1 \cup C_2) = \{Image(s_i), Image(t_i) | 1 \leq i \leq l\}$.

Let P be a path going from point a to point b in Σ. Let θ be the total angle turned through (measured counterclockwise) 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 $\theta(P) = \frac{l}{2\pi}\theta$. If $P_1, P_2, \cdots, P_k$ are non-crossing paths in G, then clearly $\theta(P_1), \theta(P_2), \cdots, \theta(P_l)$ are all equal to the same integral multiple of l.

The following lemma holds. We denote by $length(P)$ the length of path P.

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

$$\theta(P_i') - \theta(P_1^*) = \begin{cases} l & \text{if } \theta(P_i) - \theta(P_1^*) \geq 2l \\ -l & \text{if } \theta(P_i) - \theta(P_1^*) \leq -2l. \end{cases}$$

Proof: Assume that $\theta(P_i) - \theta(P_1^*) = rl$ for an integer $r \geq 2$: a proof for the other case is similar to one for this case. Every intersecting vertex v of paths P_i and P_1^* satisfies

$$\theta(P_i[s_i, v]) - \theta(P_1^*[s_1, v]) = l - (i-1) \bmod l$$

and

$$\theta(P_i[v,t_i]) - \theta(P_1^*[v,t_1]) = i-1 \bmod l.$$

Assume that the intersecting vertices $v_1, v_2, \cdots, v_q$ of P_i and P_1^* appear in this order on P_i going from s_i to t_i. Define r_x, $1 \le x \le q$: $r_x = \frac{1}{l}\{\theta(P_i[s_i,v_x]) - \theta(P_1^*[s_1,v_x]) + i - 1\}$. Then the sequence of integers $r_1, r_2, \cdots, r_q$ satisfies

$$\begin{cases} r_1 = 0,\ 1; \\ r_q = r, r+1; \text{ and} \\ r_x - r_{x+1} = 0, \pm 1, \text{ for every } x,\ 1 \le x \le q-1. \end{cases}$$

We prove this lemma separating into the following two cases.

Case 1: $r_q = r$.

In this case, $\theta(P_i[v_q,t_i]) - \theta(P_1^*[v_q,t_1]) = i-1$. There exists an intersecting vertex v_a of P_i and P_1^* such that

i) $r_a = 1$; and

ii) $r_x \ge 1$ for every x, $a \le x \le q$.

Especially, let v_a be the vertex closest to v_q on $P_1^*[s_1,v_q]$ among such vertices. Then we claim that v_a satisfies

iii) $P_i[s_i,v_a]$ and $P_1^*([v_a,v_q])$ intersect only at v_a.

Assume for a contradiction that $P_i[s_i,v_a]$ and $P_1^*[v_a,v_q]$ would intersect at a vertex $v_b \neq v_a$. Let v_b be the intersecting vertex that appears first on P_i going from v_a to s_i (See Figure 6). Let C be the cycle consisting of $P_1^*[v_a,v_b]$ and $P_i[v_b,v_a]$. (In Figure 6, C is drawn in thick line.) Since v_q lies in C, $P_i[v_a,v_q]$ intersects C at a vertex $v_c \neq v_a$. Of course, v_c is on $P_1^*[v_a,v_b]$ and $1 \le c \le q$. Choose as v_c the intersecting vertex that appears first on $P_i[v_a,v_q]$ going from v_a to v_q. Then $r_c = 1$ because $P_i[v_a,v_c] + P_1^*[v_a,v_c]$ is a simple cycle and $\theta(P_i[v_a,v_c]) = \theta(P_1^*[v_a,v_c])$. Of course, $r_x \ge 1$ for every x, $c \le x \le q$. However, v_c is closer to v_q than v_a on $P_1^*[s_1,v_q]$, contradicting the selection of v_a.

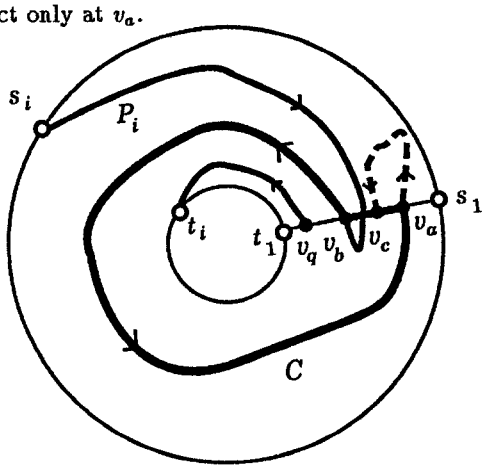

Fig.6. Illustration for the proof of Lemma 2.

Let $P_i' = P_i[s_i,v_a] + P_1^*[v_a,v_q] + P_i[v_q,t_i]$, then P_i' is a simple path connecting s_i and t_i and $\theta(P_i') - \theta(P_1^*) = l$. Since $length(P_1^*[v_a,v_q]) \le length(P_i[v_a,v_q])$, we have $length(P_i') \le length(P_i)$.

Case 2: $r_q = r+1$.

In this case $\theta(P_i[v_q,t_i]) - \theta(P_1^*[v_q,t_1]) = (i-1) - l$. We can prove that there exists an intersecting vertex v_a of P_i and P_1^* such that

i) $r_a = 2$;

ii) $r_x \ge 2$ for every x, $a \le x \le q$; and

iii) $P_i[s_i,v_a]$ and $P_1^*[v_a,v_q]$ intersects only at v_a.

Then $P_i' = P_i[s_i,v_a] + P_1^*[v_a,v_q] + P_i[v_q,t_i]$ is a simple path connecting s_i and t_i such that $\theta(P_i') - \theta(P_1^*) = l$ and $length(P_i') \le length(P_i)$. Q.E.D.

LEMMA 3. Let $\theta = 0, \pm l, \pm 2l, \cdots$, and let P_1 be a shortest path among ones which connects s_1 and t_1 and have angle θ. Then G contains paths $P_2, P_3, \cdots, P_l$ such that

(a) $\theta(P_2) = \theta(P_3) = \cdots = \theta(P_l) = \theta$, and $P_1, P_2, \cdots, P_l$ are non-crossing; and

(b) each P_i, $2 \leq i \leq l$, is a shortest path among ones which connect s_i and t_i and have angle θ.

From Lemmas 2 and 3 we have the following lemma.

LEMMA 4. Let P_1^* be an arbitrary shortest path connecting s_1 and t_1 in G. Then G contains shortest non-crossing paths $P_1, P_2, \cdots, P_k$ such that $\theta(P_1) - \theta(P_1^*)$ is either 0, l, or $-l$.

Let P_1^* be a shortest path between s_1 and t_1 in G, and let G_0' be the slit graph of G for P_1^*. 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. Denote by P_1^+ and P_1^- the two paths in G corresponding to the shortest paths in G_0' between v' and w'' and between v'' and w', respectively. Clearly $\theta(P_1^+) - \theta(P_1^*) = l$ and $\theta(P_1^-) - \theta(P_1^*) = -l$. In Figure 7 P_1^+ and P_1^- are drawn in solid and dotted lines, respectively. From Lemma 4 we have the following lemma.

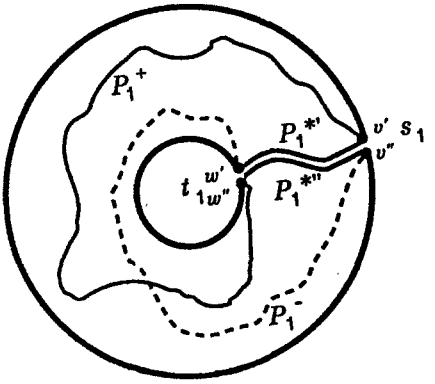

Fig.7. Slit graph G_0'.

LEMMA 5. There exist shortest non-crossing paths $P_1, P_2, \cdots, P_k$ such that P_1 is either P_1^*, P_1^+ or P_1^-.

The discussion above leads to the following algorithm.

procedure PATH(G);
begin
1. find a shortest path P_1^* between s_1 and t_1 in G;
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 $\mathcal{P}_i, 0 \leq i \leq 2$, becomes a set of non-crossing paths}
4. construct the slit graph $G_1' = G(P_1^+)$ and the slit graph $G_2' = G(P_1^-)$;
5. **for** $i := 0$ **to** 2 **do**
   begin
6. PATH2($G_i', S - (s_1, t_1)$);
7. add the found paths $P_2, P_3, \cdots, P_k$ 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 1. Given a plane graph G with k terminal pairs on its two face boundaries, one can find shortest non-crossing paths 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, spends time $T(n) = O(n \log n)$ for a plane graph [AHU,Tar]. On the other hand Frederickson's method spends time $T(n) = O(n)$ with preprocessing time $O(n\sqrt{\log n})$ [Fre]. Therefore our algorithm can be done in time $O(n(\sqrt{\log n} + \log k)) = O(n \log n)$.

5. Conclusion

In this paper, we presented an efficient algorithm for finding shortest non-crossing paths for the case when k terminal pairs are located on two specified face boundaries of a plane graph, and proved that the running time is $O(n \log n)$. Our algorithm works well even for the case where edge length may be negative if there are no negative cycles. We are now extending the algorithm to a more general case where terminals lie on three or more face boundaries.

Acknowledgement. This research is partly supported by Grant in Aid for Scientific Research of the Ministry of Education, Science, and Culture of Japan under a grant number: General Research (C) 04650300.

References

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

[Fre] G. N. Frederickson, Fast algorithms for shortest path in planar graphs, with applications, SIAM J. Comput., 16, pp. 1004-1022 (1987).

[GT] 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, pp. 209-221 (1985).

[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, the Netherlands (1982).

[Lyn] J. F. Lynch, The equivalence of theorem proving and the interconnection problem, ACM SIGDA, the Netherlands (1982).

[SAN] H. Suzuki, T. Akama and T. Nishizeki, Finding Steiner forests in planar graphs, Proc. of First SIAM-ACM SODA, pp. 444-453 (1990).

[Tar] R. E. Tarjan, Data Structures and Network Algorithms, SIAM,Philadelphia, PA (1983).